REPORT NO. NADC-82183-50

A SOFTWARE ENGINEERING ENVIRONMENT FOR WEAPON SYSTEM SOFTWARE: FUNCTIONAL DESCRIPTION FOR THE CODE AND TEST PHASE

George Mebus
Software and Computer Directorate
NAVAL AIR DEVELOPMENT CENTER
Warminster, Pennsylvania 18974

NADC
30 NOVEMBER 1982
Tech. Info.

PHASE REPORT

Approved For Public Release; Distribution Unlimited

Prepared for
NAVAL AIR SYSTEMS COMMAND
Department of the Navy
Washington, DC 20361

8500361

NOT FOUND

REPORT NO. NADC-82183-50

A SOFTWARE ENGINEERING ENVIRONMENT FOR WEAPON SYSTEM SOFTWARE: FUNCTIONAL DESCRIPTION FOR THE CODE AND TEST PHASE

George Mebus
Software and Computer Directorate
NAVAL AIR DEVELOPMENT CENTER
Warminster, Pennsylvania 18974

30 NOVEMBER 1982

PHASE REPORT

Approved For Public Release; Distribution Unlimited

Prepared for
NAVAL AIR SYSTEMS COMMAND
Department of the Navy
Washington, DC 20361

N O T I C E S

REPORT NUMBERING SYSTEM - The numbering of technical project reports issued by the Naval Air Development Center is arranged for specific identification purposes. Each number consists of the Center acronym, the calendar year in which the number was assigned, the sequence number of the report within the specific calendar year, and the official 2-digit correspondence code of the Command Office or the Functional Directorate responsible for the report. For example: Report No. NADC-78015-20 indicates the fifteenth Center report for the year 1978, and prepared by the Systems Directorate. The numerical codes are as follows:

CODE	OFFICE OR DIRECTORATE
00	Commander, Naval Air Development Center
01	Technical Director, Naval Air Development Center
02	Comptroller
10	Directorate Command Projects
20	Systems Directorate
30	Sensors & Avionics Technology Directorate
40	Communication & Navigation Technology Directorate
50	Software Computer Directorate
60	Aircraft & Crew Systems Technology Directorate
70	Planning Assessment Resources
80	Engineering Support Group

PRODUCT ENDORSEMENT - The discussion or instructions concerning commercial products herein do not constitute an endorsement by the Government nor do they convey or imply the license or right to use such products.

APPROVED BY: [illegible] DATE: 4/12/83

TABLE OF CONTENTS

A Software Engineering Environment for Weapon System Software: Functional Description for the Code and Test Phase

George Mebus

Software and Computer Directorate
NAVAL AIR DEVELOPMENT CENTER
Warminster, Pennsylvania 18974

1. Purposes of a SEE

"Software Engineering" is concerned with developing software systems that satisfy the requirements of the user over the whole life of the system; a SEE assists the accomplishment of software engineering through sets of computer facilities, integrated software tools, and uniform engineering procedures. The term "weapon system software" inherently implies a concern with software for embedded computer systems and support over the entire life-cycle. Among the aims of a SEE are to increase productivity, support development of reproducible software and promote standardization of software and software development methods. The industrialization of computer hardware is already successful in these areas; the SEE will enable a similar industrialization of computer software.

Productivity

To increase productivity, the SEE must let users work in the most productive mental surroundings, i.e., at a high level of abstraction, dealing with the problem they are to solve, rather than the details of the host computer system. DoD recognized this when they emphasized High Order Languages (HOL's) and Ada[1] in particular. The SEE must automate what can be automated to free users of details irrelevant to their work (how to access the compiler, organization of file structures, etc.), and guide users in systematic software development.

This guidance will both disencumber users and restrict what they can do. Structured programming provides a taste of this freedom with restriction. It makes one's thinking about decisions easier by providing a standard set of control structures. It also restricts use of a large class of possible control statements because they may adversely affect program quality. Thus, productivity is enhanced by both the provisions and the restrictions of structured programming. The SEE's guidance will have similar benefits.

Software sharing is also essential to increased productivity. Repeated re-invention of software is a costly waste of resources. Software developers should be able to make general, flexible

[1]Ada is a Registered Trademark of the Ada Joint Program Office - U.S. Government

subprograms that can be freely shared. The programming language used greatly affects the generality of software modules. But the software engineering environment also affects the ability to know about the software components as well as the freedom to access them. SEE will provide convenient support of software sharing.

Reproducibility

The Navy must be able to reproduce its software. This means that the Navy must have control over the tools used to generate the software. A contractor's proprietary tools must not be part of the software development process because they tend to lock the contractor into the project for the entire life cycle. Also, the Navy must limit users' ability to make arbitrary changes to their data base information. For example, if someone can modify the object code so that it does not reflect the associated source code (a current capability of the MTASS SYSGEN) then the Navy cannot always reproduce the object code from the source.

Standardization

Standardization is essential to the above topics. It is also necessary for automated Configuration Management. The SEE will enforce both internal standards (file formats, file contents, file and tool accesses, development data collection, etc.) and external standards (e.g., compliance with MIL-STD-1679).

Standardization carries much weight in the aims of Ada. Ada is intended to be a standard language across all DoD with neither subsets nor dialects. A standard language support system is also envisioned. Proposed benefits of these standardizations include user transportability, software transportability and reduced re-invention. SEE standardizations have similar virtues to promote higher productivity and better software.

One important feature of the SEE is the standard data base structure, automatically provided and maintained. A typical benefit is the transition of software from software developer to the Software Support Activity (SSA). Although usually a tedious, expensive process it becomes trivial when both developer and SSA have the software in standardized SEE data bases throughout the different life cycle phases.

2. Present Navy Efforts

MTASS

The Machine Transportable AN/UYK-20, AN/AYK-14 Support Software (MTASS) is a set of tools for military software development, hosted on several large commercial computers and targeted for the military computers mentioned in its name as well as the AN/UYK-44. The tools are cooperative in that the outputs of some are the inputs of others. They are standardized in that they work the same way on the various hosts and they provide a basic set of tools for all programmers of the target machines. The set is not complete in that some use of auxiliary host machine tools is required (e.g., for source and object library maintenance). PMS-408 maintains the MTASS source code and distributes only the object.

SHARE/7

An interactive operating system for the AN/UYK-7 military computer. It allows direct user manipulation of files, tapes and tools. While the Navy controls who may use SHARE/7, there is no Navy control over its use, the content or structure of files or the methods of tool invocation.

UNIX[2]

An interactive operating system for DEC VAX[3] computers and other commercial hosts. UNIX has an extensive compatible tool set, a flexible convenient file system and good internal communication among tools. Most tools do a simple operation on a file, producing another file as a result. Many desirable file processing operations can be quickly developed by stringing various tools together. UNIX provides a tree-structured file system with flexible ways of accessing and processing groups of files. Since UNIX is a general purpose commercial operating system, the Navy does not control its distribution and use.

FASP

A Navy integrated software development environment, hosted on large-scale commercial computers and targeted to fully support all Navy standard military computers. FASP enforces development standards through the mandatory use of "procedures" tailored by parameters to do the required tasks. The automated procedures remove the necessity for users to directly access either tools or files. Thus they release users from the tedious intricacies of operational detail, while giving advanced programming capabilities to the user, control of the tools to the Navy, and control of the files to a project manager. The manager is also able to control the use of both procedures and parameters via a set of "keys" which establish the system privileges afforded to each user. A set of management report procedures enhance management visibility by producing reports from data automatically gathered during software development.

Two versions of FASP currently exist. One hosted on CDC CYBER machines, using KRONOS and NOS operating systems, supports many major Navy projects using the Navy standard embedded military computers (e.g., AN/AYK-14, AN/UYK-44, AN/UYS-1). Extensive tool development was done to make an integrated environment. The CYBER version of FASP has been in use for the past seven years. The second version, which supports microprocessor code development, is hosted on VAX machines

[2]UNIX is a trademark of Bell Laboratories.
[3]DEC and VAX are trademarks of Digital Equipment Corporation.

using the UNIX operating system. A much greater use of existing operating system facilities was made in UNIX than with the CYBER operating system. A faster, easier implementation was possible because the UNIX tools were more easily combined and new tools were more easily created through use of the UNIX shell and the C language. NAVAIRDEVCEN is the developer and maintainer of FASP.

Ada MAPSE

The Minimum Ada Program Support Environment (MAPSE) will be an environment for military software development. The goals are to have a transportable tool set, good inter-tool communication, and standard tools for all users. STONEMAN defines requirements of the MAPSE. The Ada Program Support Environment (APSE) is less clearly defined although broad principles and goals exist. PMS-408 will control the Navy Ada MAPSE as it does MTASS.

3. Distinctives of the SEE

Two notable areas of the SEE are its technical characteristics and user-friendliness. Technical characteristics are design features of the SEE, while user-friendliness is an intended by-product of them.

Technical Characteristics

Multiple language support is essential for the SEE. Even when Ada is firmly established, the SEE must still maintain existing fleet software written in a variety of current Navy-standard languages. Differences in compiler facilities, object libraries, and target computer emulators require different tool sets for each supported language. The SEE will offer standardized facilities for dealing with the several language systems to provide a consistent framework for software maintenance.

Procedures perform specific high-level jobs important to software life cycle phases. Procedures are predefined sets of host computer commands that do the software development jobs by ensuring that appropriate tools are properly sequenced and invoked, that appropriate files are properly accessed and created, that development information is collected and saved, etc. The SEE may develop and execute hundreds of operating system statements in doing a single procedure. Procedures are invoked by commands whose names reflect the jobs they do (e.g., Develop Software, Execute Tests). Mnemonic abbreviations are available for convenience (e.g., DEVSW, EXTEST). The procedure operations are tailored by parameters (like Translator or its abbreviation XLATR), most of which have default values (like XLATR=ADA). These parameter values add the flexibility required by specific tasks. A procedure handles all complexities of tool invocation and sequencing, data base management, error recovery, and collection of development statistics, with no need for the users' involvement or awareness.

Encapsulated data bases are standard sets of files with standard organizations. For a set of software modules a data base contains source, object, tests, test results, development histories, documentation, etc. With a fixed structure that users cannot directly change, the SEE assures that the proper information in the proper relationship is available for Configuration Management (CM) reporting, management tracking and re-creation of software. It can check the internal integrity of a data base. Regularity of the data base results in efficient processing and also allows efficient enhancement of the SEE in that any new language-independent procedure can be applied to any SEE data base, thus reducing development cost.

User-friendliness

Programmers are lifted above internal details of the host operating system and repetitive operations. Some details are unrelated to program design and coding (such as compiler access or object library directives) and are prone to error. Others may even be dangerous. If unrestricted access to files were allowed, users could independently modify any file, potentially destroying file relationships. The SEE assures that tools are accessed and sequenced properly, requiring users to provide only their program data as inputs.

Managers benefit in two areas: visibility and control. Visibility is presented through a series of software and CM reporting procedures. They tell what software has been developed, the current testing activity, how development is tracking previous estimates, etc.

The manager initially exercises control when creating new data bases or defining "keys" for programmers. Subsequent control is a result of the key definitions. For example, a manager may not want a new programmer to be able to modify a data base but only make temporary changes for the present. The keys may then prevent use of modifying procedures like Develop Software, but allow procedures that produce reports. Similarly, the manager can restrict a programmer from using an particular version of a translator by restricting the use of the Translator parameter, forcing the default value to be used when software is modified.

4. Purposes of the Software Functional Description

Appendix A presents the Software Functional Description of a SEE. The design is general and applicable to all Navy software development. The Functional Description concentrates on the code and test phases but the adequacy and advantages of its organization clearly apply to earlier phases as well.

Discussing a complex system like a SEE often becomes a confusion of details and specifications that cloud the power and usefulness of the capabilities the system affords. For clarity this Software Functional Description shows the salient features of a SEE at a fairly high level--the user interface. The features are organized by user activities (e.g., Develop Software, Execute Tests, Document). Each activity has an associated "procedure" in the SEE and an associated command name to invoke the procedure.

The Functional Description is arranged as a representative list of procedures corresponding to the discrete steps performed by programmers and software managers. In itself this list demonstrates how the SEE helps to organize user activity. Each procedure is individually presented in the Functional Description in several ways. First, a narrative text verbally describes the intent of a procedure and the capability it provides. Next, the appropriate parameters are listed to show how the users can control the operations performed. The parameter list also shows the level of detail users must consider.

After this a graphic display shows what goes on "behind the scenes" from the viewpoint of software architecture. This graphic display uses structured English to define the flow of control. The control flow description is coupled with a parallel pictorial diagram that defines the data flow. These two parallel flow descriptions illustrate several things:

- the many automated steps executed by the procedure in response to the single command invocation;

- just how parameters affect the processing and provide the needed flexibility;

- data flow information, data base contents used and produced, and other necessary file information;

- the protection given to processing, tools and data base contents;

- automatic provisions for error recovery.

In all, the Software Functional Description shows the degree to which software development can be automated and indicates the complexity that users need not contend with, but which is instead designed into the SEE. It is intended to give a clear picture of the operation of a SEE that meets needs of software developers, managers and the Navy.

APPENDIX A

SOFTWARE FUNCTIONAL DESCRIPTION

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

FIGURES

1. INTRODUCTION

Reference (a), "The Industrialization of Weapon System Software," advances the concept of a Software Engineering Environment (SEE) composed of a Software Production Facility (SPF) and a set of hardware/software Integration Facilities (IFs). The SPF is used to develop, test and maintain operational software which is then sent to the IFs for integration. Functionally, the SPF provides named "procedures" as the first level of user communication. These procedures automatically invoke specific software tools to perform the prescribed sequences of operations. They use a Data Base Management System (DBMS) to manage the storage and retrieval of data developed during the operations.

As described in reference (a), the SPF provides three phases of software support: requirements, design, and code & test. This software Functional Description identifies the functional groups of procedures provided by the SPF. For each group, narrative descriptions and process flow diagrams of the procedures are given followed by a list of the procedures, the tools they invoke, and the data base contents used and produced by the tools.

2. REFERENCE DOCUMENTS

(a) "The Industrialization of Weapon System Software," H. G. Stuebing, 15 May, 1981.

3. ABBREVIATIONS

DB	Data Base
DBMS	Data Base Management System
FD	Functional Description
IF	Integration Facility
INST	Instrumented (resulting from software probes inserted in source code)
MTASS	Machine Transportable AN/UYK-20, AN/AYK-14 Support Software
SCP	Software Change Proposal
SEE	Software Engineering Environment
SEP	Software Enhancement Proposal
SPF	Software Production Facility
STR	Software Trouble Report
SW	Software

4. PROCEDURES, COMMANDS AND PROCESSING

The SPF is driven by procedures which are invoked by simple user commands. This system provides:

- Automated standards enforcement
- Centralized data collection
- Convenience and flexibility for the software developer
- Control, security and visibility for the software manager

4.1 Procedures

A procedure is a set of SPF computer directives which:

- Automates a particular software production task
- Invokes the proper tools in the proper sequence
- Handles all file manipulations and correspondences
- Automatically records job statistics

Procedures are available to meet all user needs (e.g., specify, design, develop, test, maintain, document, track) throughout the software life cycle. They require only information and data pertinent to the user's specific software production tasks (not tool invocations and file processing), thus simplifying the user's work.

A procedure may perform many smaller tasks, handling all details of data collection and tool invocation in the process. For example, the DEVELOP SOFTWARE procedure invokes an editor for text development, invokes the user-specified translator (e.g., CMS-2M compiler) to translate or retranslate all affected software modules, and invokes an object code interface extractor to identify all externals. The source and object code are always kept synchronized and job statistics are automatically collected for project tracking.

4.2 Software Tools

The software tools available through the SPF cover the full range of functions for mission software design, development, and maintenance. Examples are:

requirements tools	system generators
design tools	analyzers
editors	data extractors
translators	report generators

4.3 Data Bases

SPF procedures use a Data Base Management System (DBMS) to manage the storage and retrieval of data developed during SPF usage. The DBMS creates and maintains a hierarchical project tree file structure in which data bases are terminal nodes (or "leaves") of the tree. Project managers configure the project tree to reflect the structure of their projects. Figure 1 shows such a tree structure for a typical Navy software project.

Data bases are the repositories for the software and all information related to the software. A particular data base is assigned to developing and maintaining software for a particular operational capability. A data base will contain:

- requirements specifications and documentation
- design specifications and documentation
- source and object code for one or more modules
- system generator directives
- load modules (executable programs)
- tests and test results
- histories - items, changes, metrics
- configuration management information
- interface data - source and text inclusions, subroutine calls, external definitions and references

4.4 Commands

Each procedure is invoked by a command. Each command is a string of characters which begins with a procedure name, to identify the procedure to be performed. The name is typically followed by parameter value specifications. These values give the user flexibility in directing the performance of the procedure to accomplish the specific function required. There are two kinds of commands:

Immediate commands are executed as soon as received and validated by the SPF.

Queued commands are collected on a command queue for later processing.

All commands are processed as described in section 4.6.

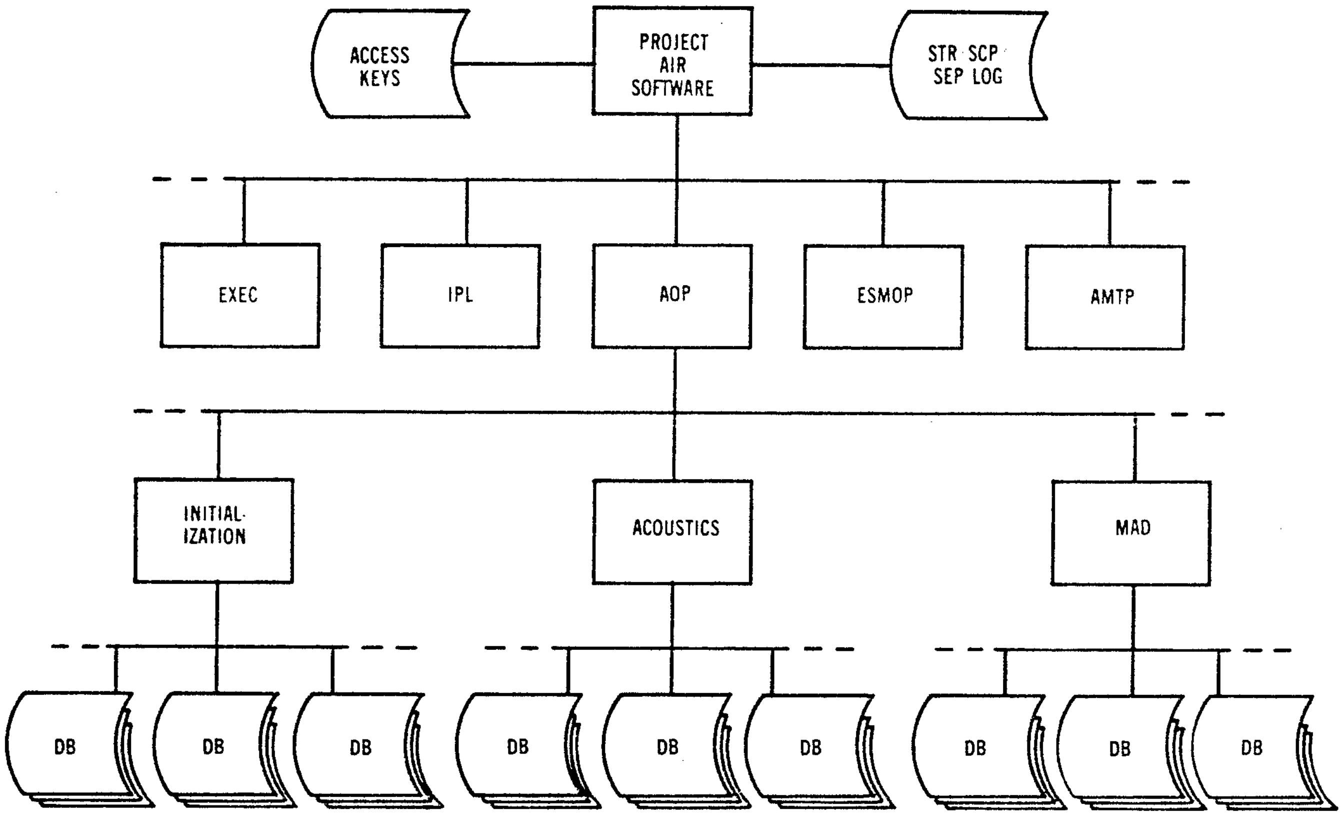


FIGURE 1
SOFTWARE PRODUCT HIERARCHY TREE

4.5 Process Flow Diagrams

Process Flow Diagrams are used in this Functional Descrition to describe the unique operation of each procedure supported by the SPF. They consist of two associated parts:

(1) a structured English description (using standard IF-THEN-ELSE-ENDIF and DOWHILE-ENDDO constructions) of the control flow for a procedure, and

(2) a data flow diagram showing process performed, tools used, data base contents used and produced, and other information required by the procedure. A legend of diagram symbols is shown in Figure 2.

When a procedure is performed, a certain amount of "standard" processing is done before and after the "unique" processing for that procedure. This standard processing done for each procedure is shown in Figure 3 using a structured English description.

4.6 Command Processing

The SPF reads and validates the user's commands. In a batch processing mode, each valid command is executed as it is received.

In an interactive mode, any command recognized as "immediate" is executed immediately. All others are placed on the command queue for later execution. Upon receipt of the EXECUTE COMMAND QUEUE command, the SPF will execute the queued commands.

Command processing is illustrated by a process flow diagram in Figure 4.

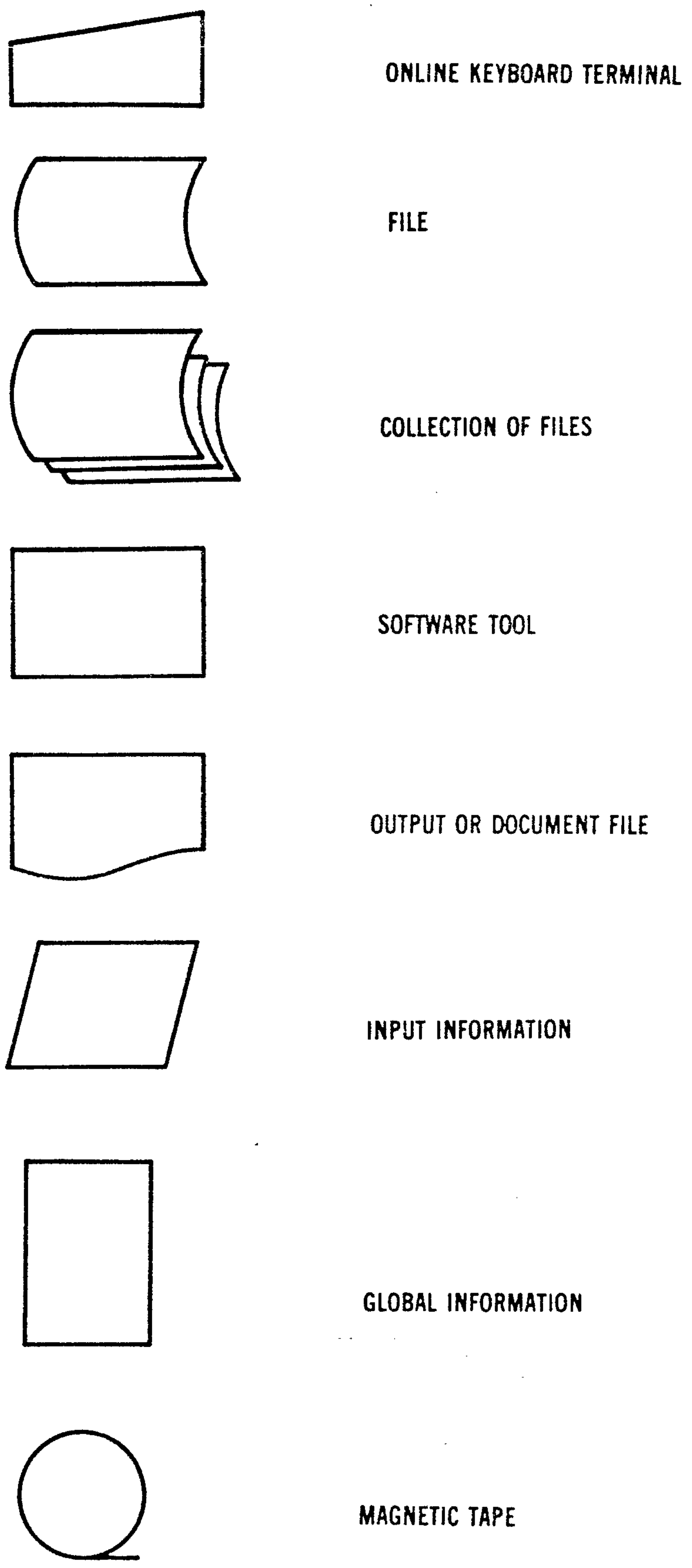


FIGURE 2
PROCESS FLOW DIAGRAM LEGEND

```
Verify user access rights for this procedure
IF verification fails
THEN raise abort flag
● Notify user
ELSE
●
●    (unique process flow for procedure)
●
ENDIF
IF abort flag is not raised
THEN save production data
● Make "saved" files permanent
ENDIF
Save job statistics
Make job statistics permanent
```

Figure 3 - Standard Processing for Procedures

```
DOWHILE no LOGOFF command received
•  DOWHILE no valid LOGON command received
•  •  Read and identify command
•  •  IF not LOGON
•  •  THEN notify user that LOGON is required
•  •  ELSE invoke LOGON processor
•  •  •  See process flow for LOGON
•  •  •  paragraph 8.4.1.1
•  •  ENDIF
•  ENDDO        (user is now identified)
•  Read and identify command
•  IF not identifiable
•  THEN notify user
•  ELSE validate command
•  •  IF error
•  •  THEN notify user
•  •  ELSE
•  •  •  Append default values of unspecified parameters
•  •  •  IF an immediate command
•  •  •  THEN  execute the command
•  •  •  ELSE put the command on the command queue
•  •  •  ENDIF
•  •  ENDIF
•  ENDIF
ENDDO
```

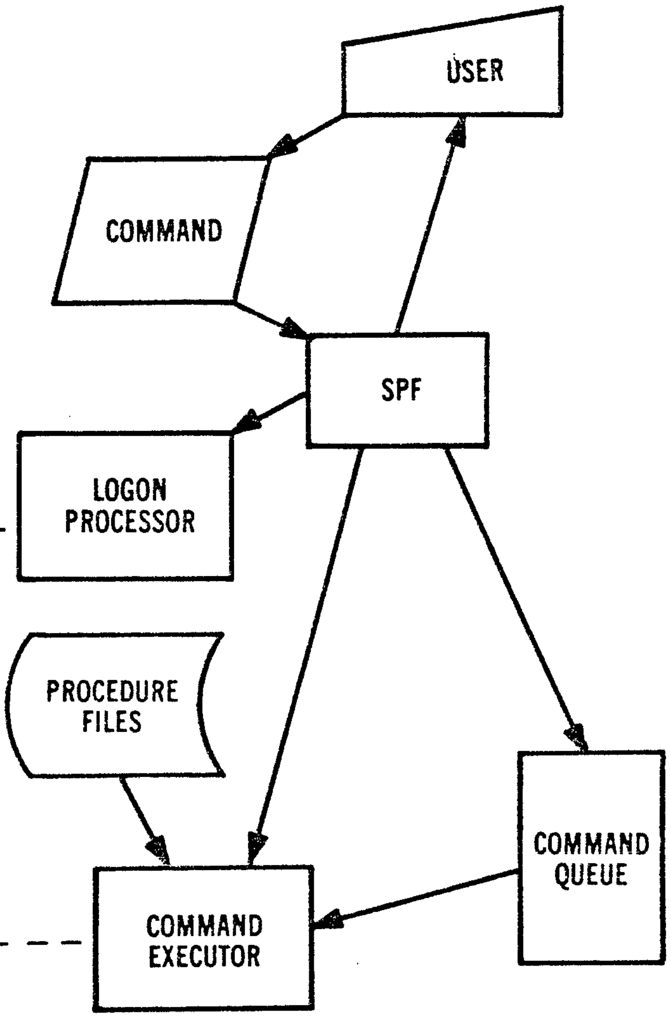


Figure 4 - Command Processing

5. SOFTWARE REQUIREMENTS SPECIFICATION AND ANALYSIS PROCEDURE GROUPS

Studies of requirements tools, particularly RSL/REVS and PSL/PSA, and their integration into a Software Engineering Environment are ongoing. Results of the studies are not available at the time of this draft.

6. SOFTWARE DESIGN PROCEDURE GROUPS

A study of software design methodologies is ongoing. The aim of the study is to identify one or a combination of methodologies which will best serve Navy Weapon System software and identify how automated tools can aid the use of that methodology. Results of the study are not available at the time of this draft.

7. SOFTWARE CODE AND TEST PROCEDURE GROUPS

7.1 Software Development Procedures

The "code" part of "code and test" is oriented toward operational software development. Developers must be able to establish a safe repository for their software, develop that software and make loadable object code for test integration and fleet issue. They must also be able to get current information about the software content and organization, and control the disposition of job output.

		DB CONTENTS	
PROCEDURES	TOOLS	USED	PRODUCED
Create/Copy/Save/ Restore DB	DBMS	File Directories/ Dependencies	File Directories/ Dependencies
Develop SW	Text Editors Translators	Source/Object Code	Source/Object Code
	Object Interface Extractors		SW Histories SW Interfaces
Install External SW	Preprocessors Translators Object Interface Extractors		Source/Object Code SW Histories SW Interfaces
Share SW		Source/Object/ Tests/Text	Source/Object/ Tests/Text
Copy SW	Source Selector Translators Object Interface Extractors	Source (and possibly Object/ SW Histories/ SW Interfaces)	Source/Object SW Histories SW Interfaces
Create Load Module/Tape	System Generator	System Generator Script Object Code	Load Module
Print Reports	Data Extractor Report Generator	Data Base Contents Source Code	

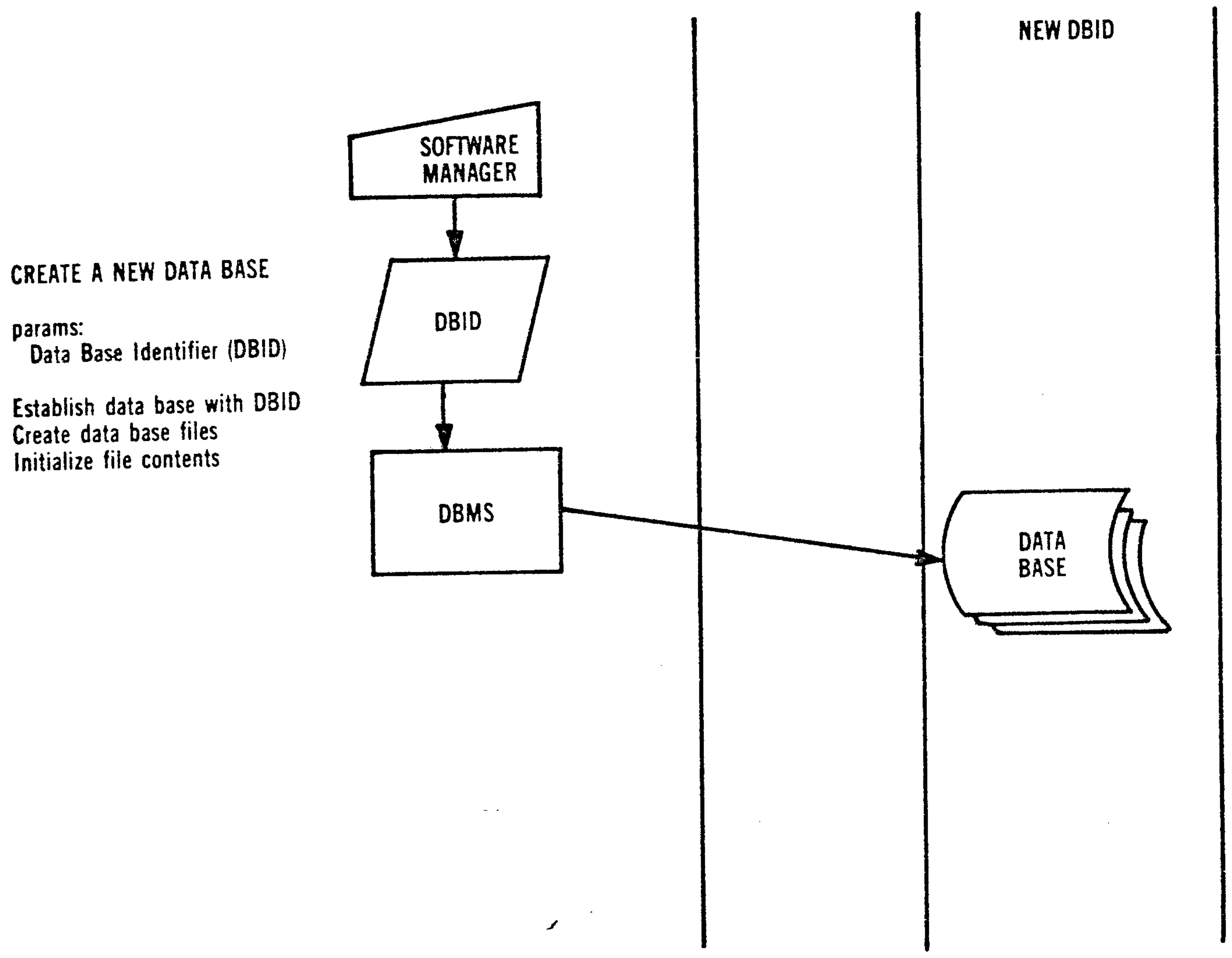
Figure 5 - Summary of Software Development Procedures

7.1.1 Create/Copy/Save/Restore a Data Base

The SPF, through the DBMS, automatically maintains each data base and the information on it. Use of these procedures is typically restricted by the software manager.

7.1.1.1 Create a Data Base

The SPF creates the files and directories comprising an initial configuration of a data base with a specified identifier at a specified project tree terminal node.



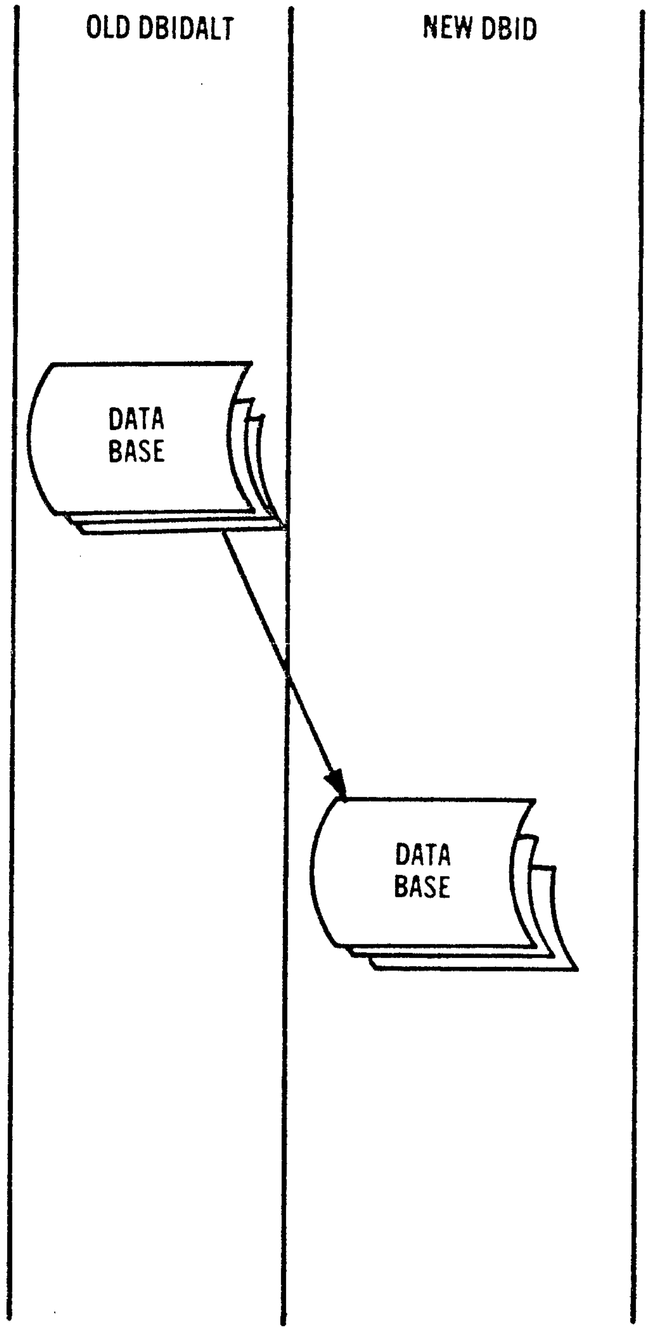
7.1.1.2 Copy a Data Base

The SPF creates a new data base at a specified node. The data base duplicates the file structure and contents of a specified alternate data base.

COPY A DATA BASE

params:
  Data Base Identifier (DBID)
  Alternate Data Base Identifier (DBIDALT)

Establish data base with DBID
Create data base files
Copy contents of DBIDALT files into DBID files



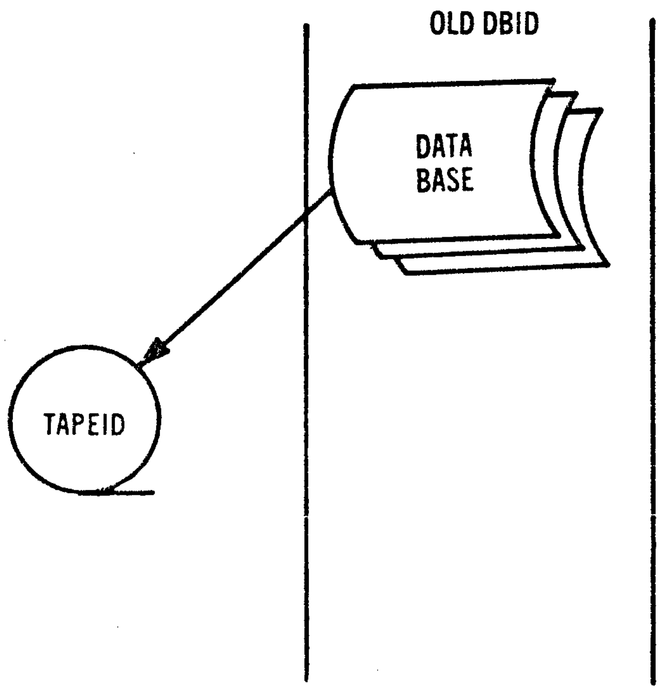
7.1.1.3 Save a Data Base

The SPF copies the specified data base structure and contents to a specified magnetic tape. A data base may be saved for baselining or for file damage protection beyond that automatically provided by the SPF.

SAVE A DATA BASE ON TAPE

params:
  Data Base Identifier (DBID)
  Tape Identifier (TAPEID)

Copy DBID files to TAPEID tape



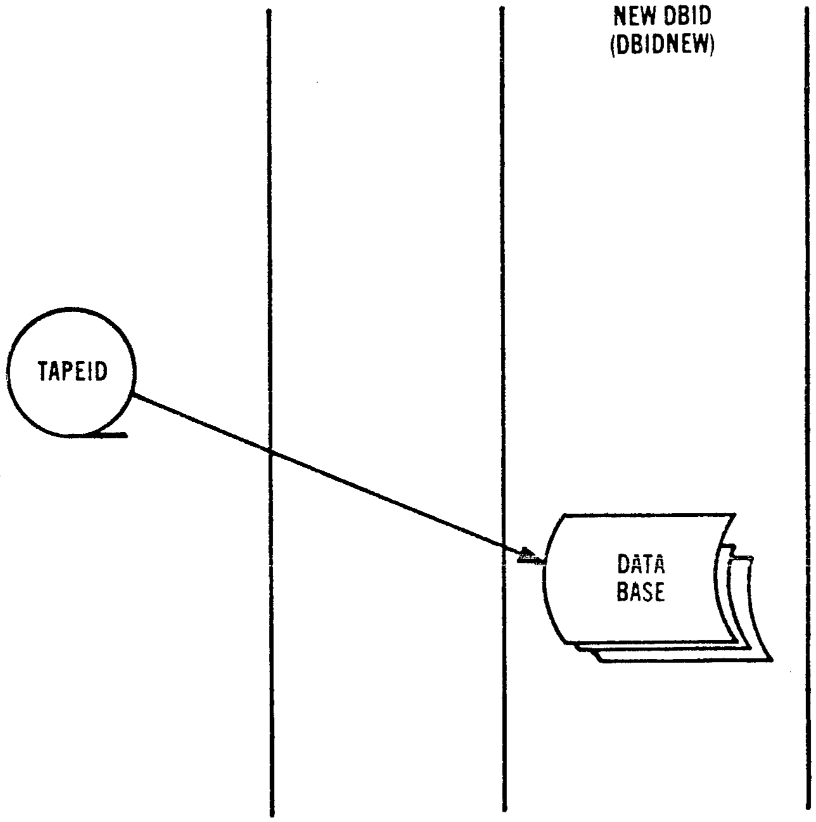
7.1.1.4 Restore a Data Base

The SPF creates a new data base and copies the contents of the specified magnetic tape into the data base, thus completing the restoration of an archived data base.

RESTORE A DATA BASE FROM TAPE

```
params:
  Data Base Identifier (DBID)
  New Data Base Identifier (DBIDNEW)
  Tape Identifier (TAPEID)

IF tape not recorded from DBID
THEN raise abort flag
ELSE establish data base with DBIDNEW
• Create data base files
• Copy TAPEID contents into DBIDNEW files
ENDIF
```



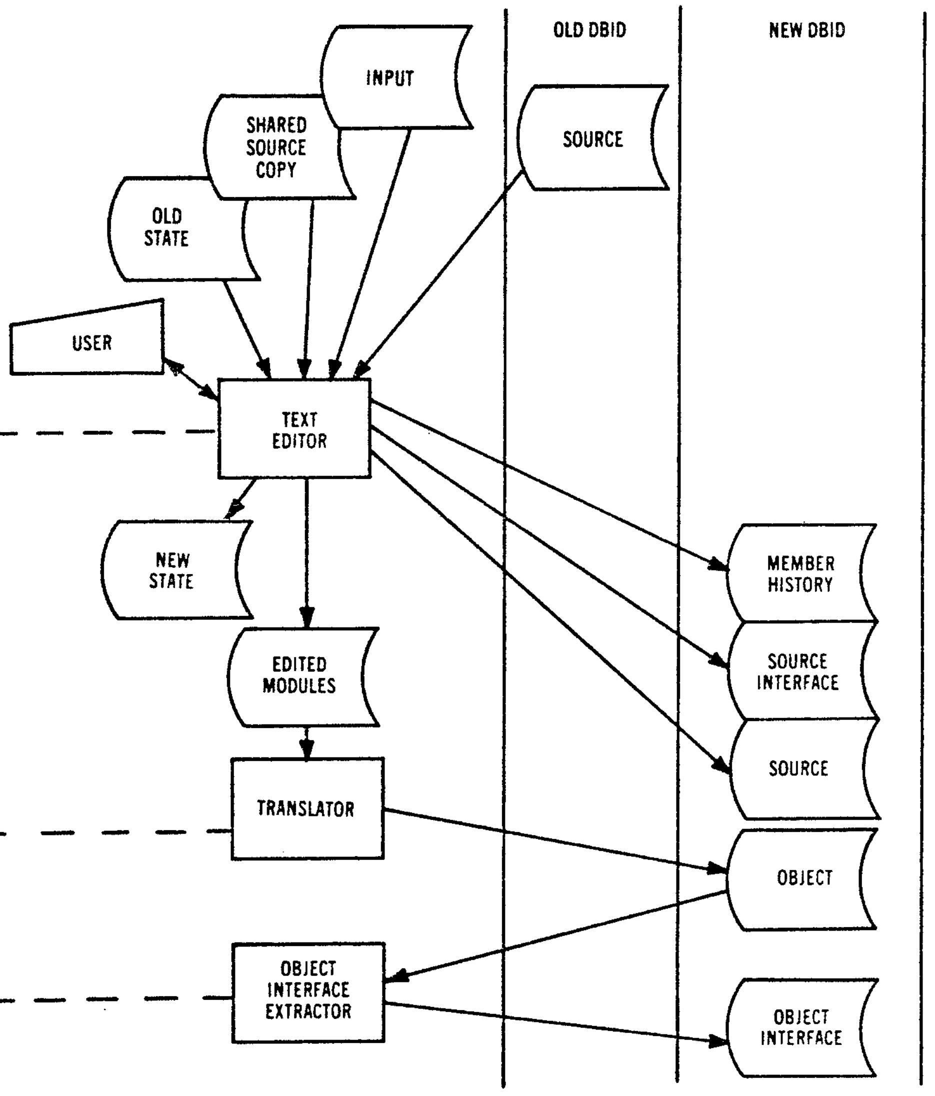
7.1.2 Develop Software

This procedure allows entry of source code into a data base and subsequent modification of source code. All source affected by entry or modification is automatically (re)translated to ensure that source and object code are always synchronized. Other information developed and saved on the data bases are software histories and software interface information (cross references, externals information, software dependencies).

```
DEVELOP SOFTWARE

params:
  Data Base Identifier (DBID)
  Input Data File (INPUT)
  Source Code Translator (TRANSLATOR)

Access INPUT file
Access source code
Access shared source copy (if there)
Invoke source code Text Editor
  IF old state file exists
  THEN identify file to user
  • IF desired
  • THEN use state file
  • ELSE clear editor state
  • • Delete the state file
  • ENDIF
  ENDIF
  Process source code from INPUT (if there)
  IF an interactive job
  THEN
  • Read source code
  • DOWHILE no END command received
  • • Process commands from the user
  • ENDDO
  ENDIF
  Create a new state file
IF included code members were modified
THEN mark all modules including them as edited
ENDIF
Build a file of all edited modules
Save member modification history information
Save source interface modification information
Save modified source code
Invoke the TRANSLATOR
  Read source code
  Produce object code
IF fatal errors
THEN raise abort flag
ELSE save the new object code
• Delete the new state file
• Invoke Object Interface Extractor
•   Read object
•   Produce object interface information
• Save object interface modification information
ENDIF
```



7.1.3 Install External Software

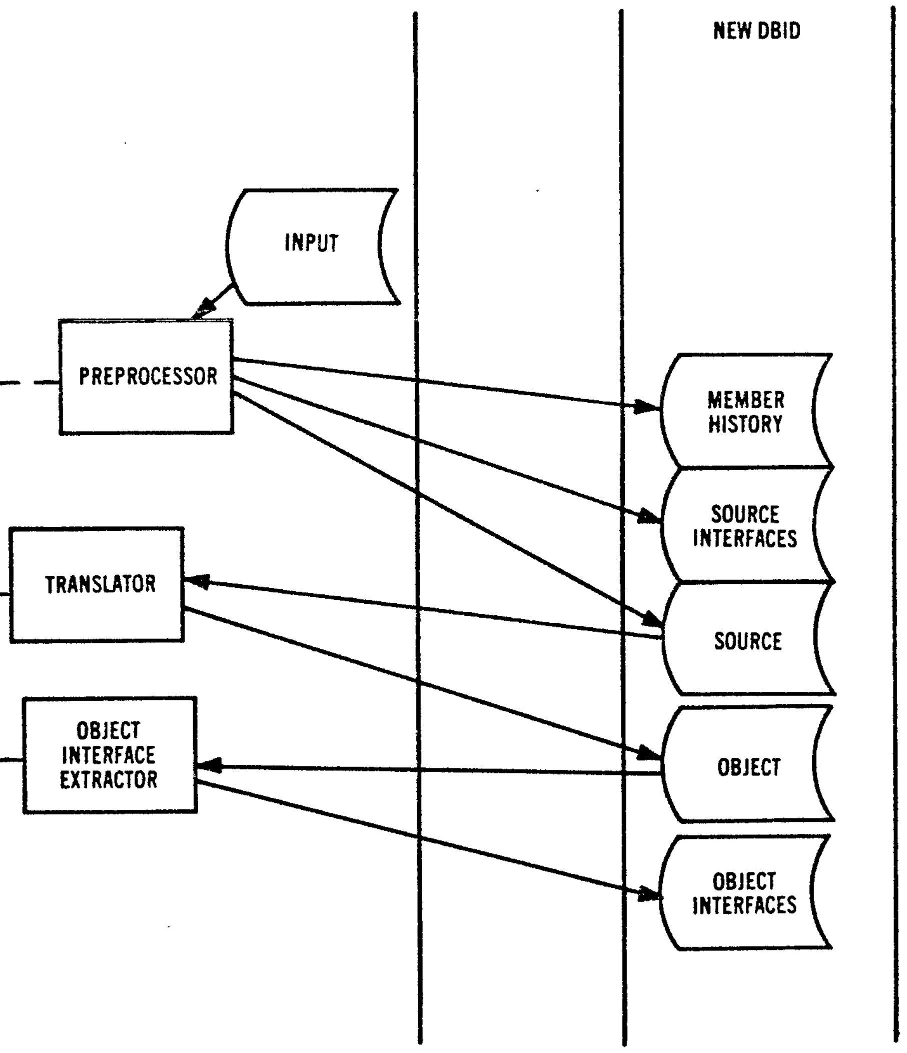
Existing software developed outside of the SPF requires additional processing when transferring it to SPF data bases. Requirements on SPF processing of external software depend on the developers adherence to the content and formats specified in reference (b). In particular, preprocessors may be required if the source code is written in a language that is not a Navy standard programming language. Conversely, if Navy standard tools were used to develop the software (e.g., MTASS), it will not only make source code transfer simpler but may also permit transfer of tests and test results to the SPF.

After source code is in acceptable SPF form in data bases, the DEVELOP SOFTWARE procedure will be used to maintain it.

```
INSTALL EXTERNAL SOFTWARE

params:
  Data Base Identifier (DBID)
  Input Source Code File (INPUT)
  Source Code Preprocessor (PREPROCESSOR)
  Source Code Translator (TRANSLATOR)

Access INPUT file (if there)
Invoke PREPROCESSOR
  | Read INPUT source code              |
  | Produce Navy-standard source code   |
IF fatal errors
THEN raise abort flag
ELSE save member history information
•  Save source code interface information
•  Save source code
•  Invoke TRANSLATOR
•   | Read source code                  |
•   | Produce relocatable object code   |
•  IF fatal errors
•  THEN raise abort flag
•  ELSE save object code
•  •  Invoke Object Interface Extractor
•  •  | Read object                          |
•  •  | Produce object interface information |
•  •  Save object code interface information
•  ENDIF
ENDIF
```



7.1.4 Share/Copy Software

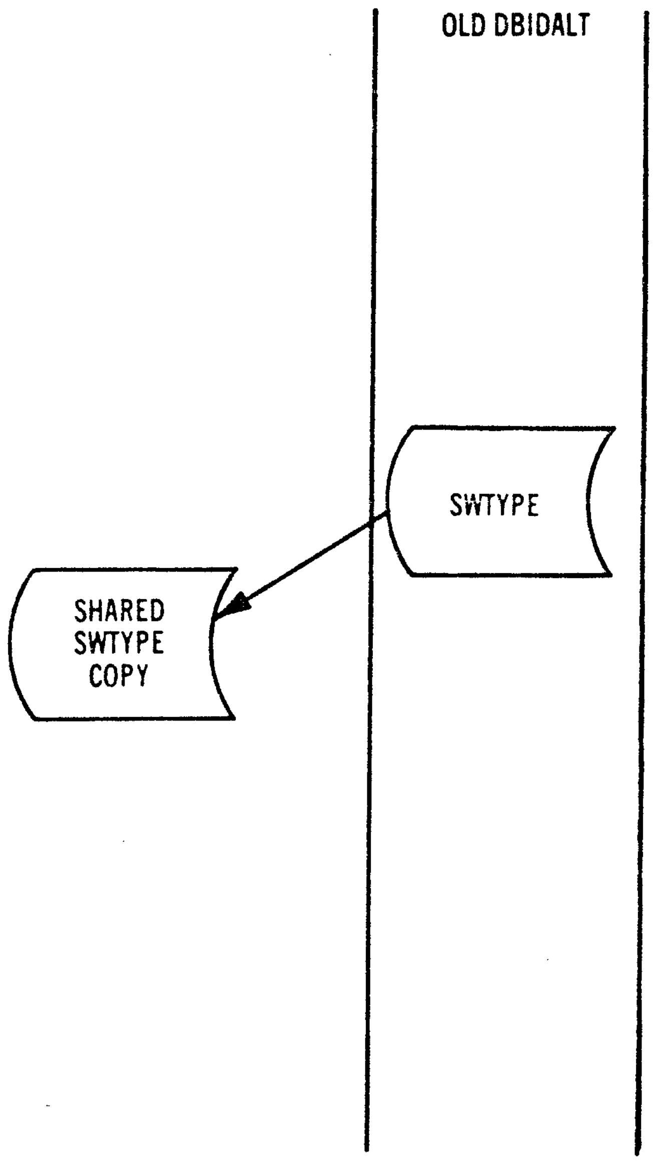
7.1.4.1 Share Software

Source code may be shared from a master data base to ensure that naming conventions and special data values are applied consistently throughout the project software. Object code, sections of tests and documentation text may also be shared among data bases.

SHARE SOFTWARE

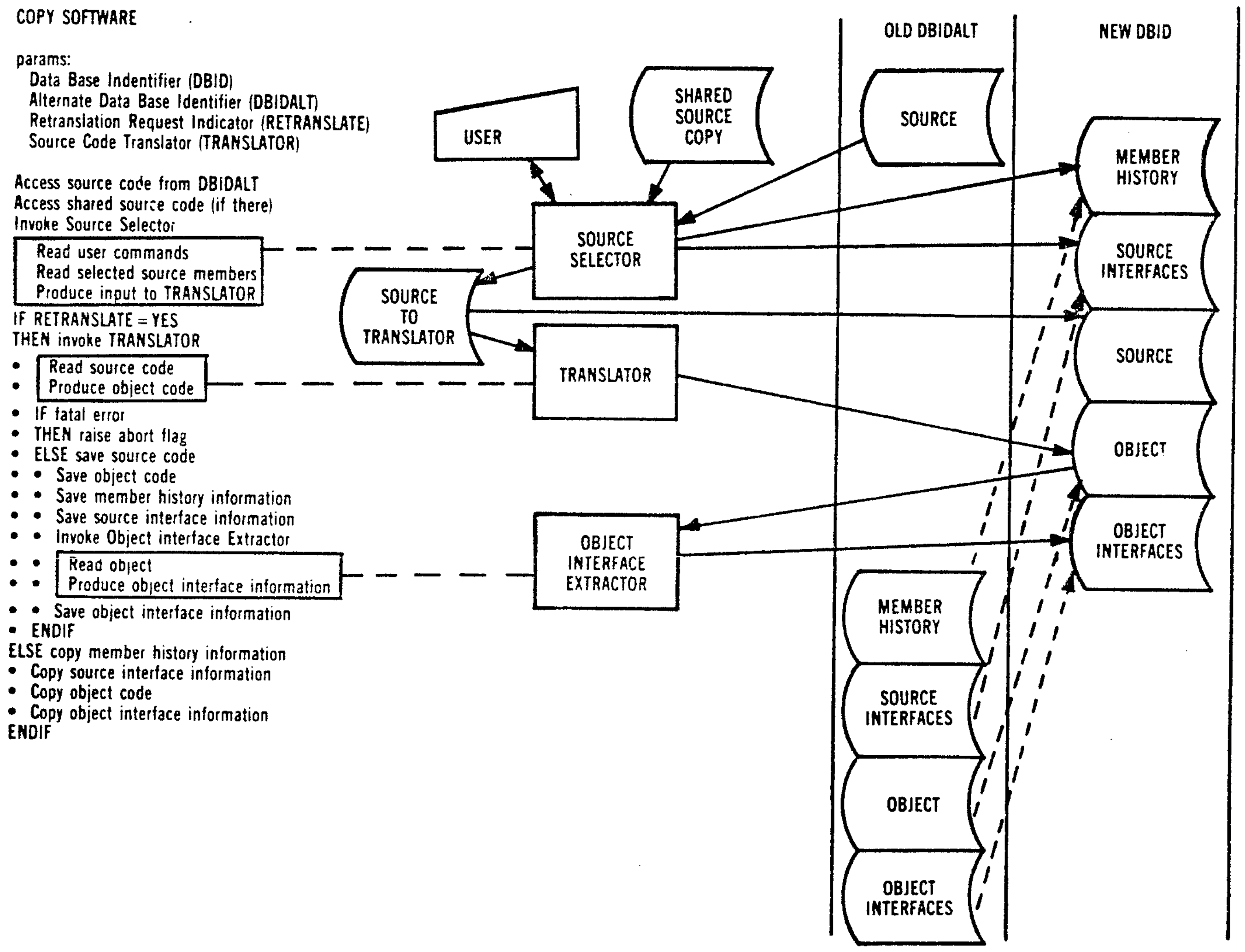
params:
  Alternate Data Base Identifier (DBIDALT)
  Software Type (SWTYPE)

Access SWTYPE file (source/object/test/text)
  from DBIDALT
Make a temporary copy for future access



7.1.4.2 Copy Software

When code is copied from one data base to another, the integrity is always maintained either by copying all pertinent information (e.g., source, object, histories, etc.) or by copying only the source code and developing the rest by translation.



```
COPY SOFTWARE

params:
  Data Base Indentifier (DBID)
  Alternate Data Base Identifier (DBIDALT)
  Retranslation Request Indicator (RETRANSLATE)
  Source Code Translator (TRANSLATOR)

Access source code from DBIDALT
Access shared source code (if there)
Invoke Source Selector
 | Read user commands               |
 | Read selected source members     |
 | Produce input to TRANSLATOR      |
IF RETRANSLATE = YES
THEN invoke TRANSLATOR
•  | Read source code     |
•  | Produce object code  |
•  IF fatal error
•  THEN raise abort flag
•  ELSE save source code
•  •  Save object code
•  •  Save member history information
•  •  Save source interface information
•  •  Invoke Object interface Extractor
•  •  | Read object                          |
•  •  | Produce object interface information |
•  •  Save object interface information
•  ENDIF
ELSE copy member history information
•  Copy source interface information
•  Copy object code
•  Copy object interface information
ENDIF
```

7.1.5 Create Load Module/Tape

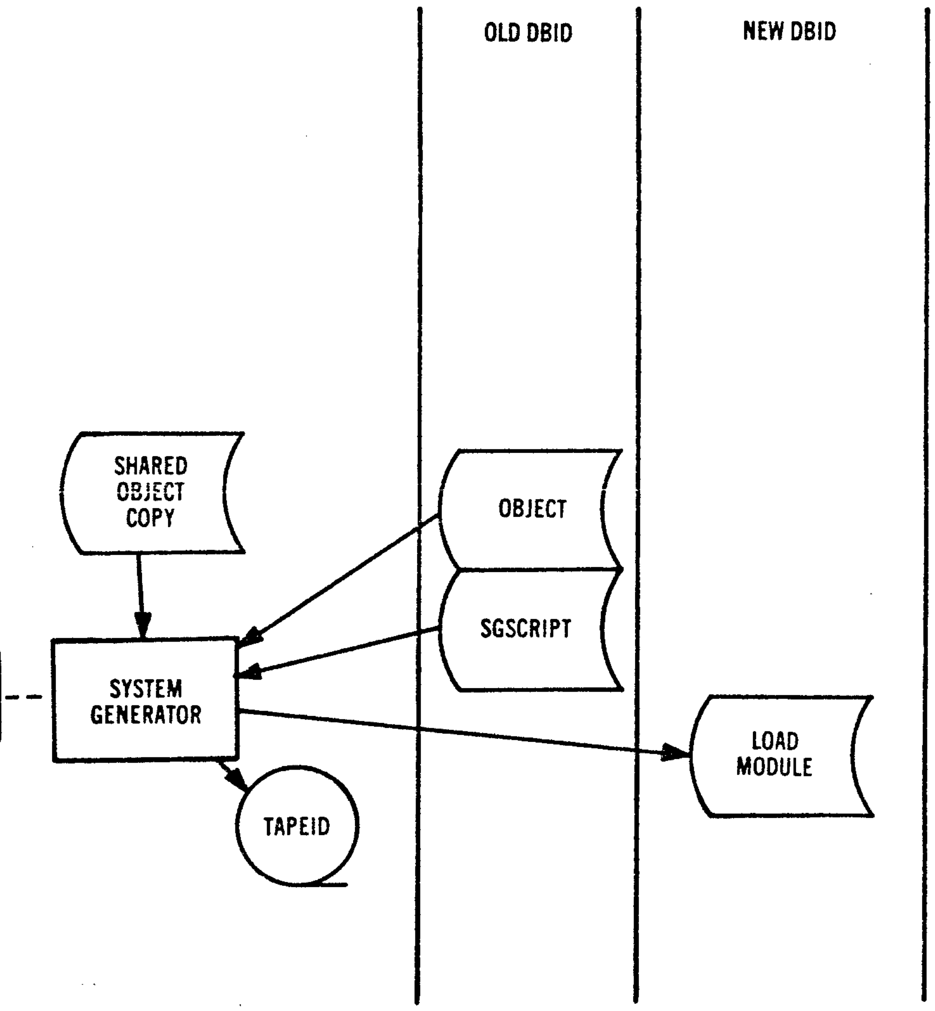
Load tapes may be created for integration or fleet issue. Load modules can be saved on a data base for: multiple tape creation, inclusion on a multiple-data base tape, or testing with a target computer emulator. Directives for a system generator are entered into and maintained on the data base. The generators use the directives and relocatable object code from the data base to create the load module file/tape.

CREATE LOAD MODULE/TAPE

params:
  Data Base Identifier (DBID)
  System Generator Script (SGSCRIPT)
  Tape Identifier (TAPEID)

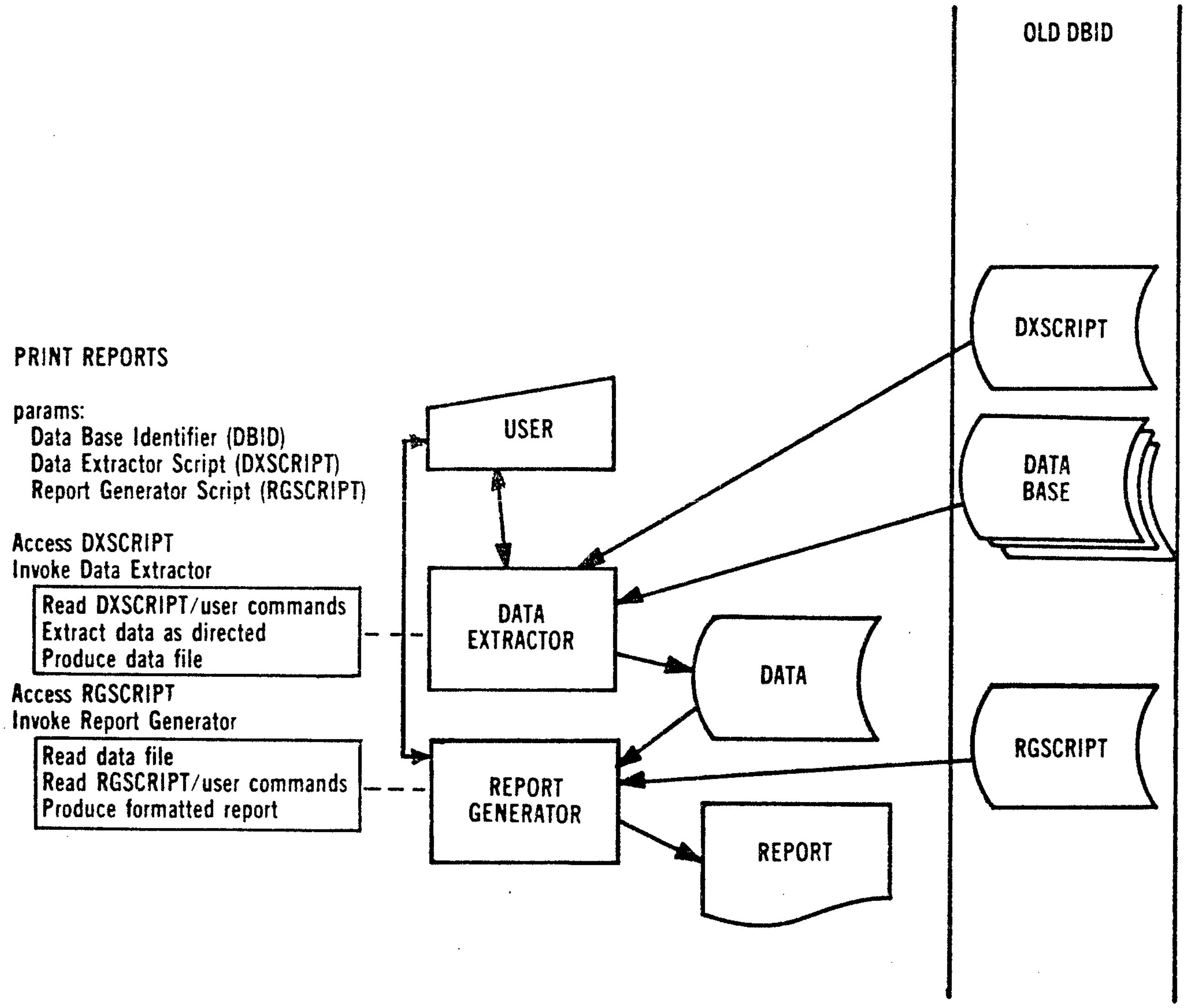
Access object code from DBID
Access shared object copy (if there)
Access SGSCRIPT
Invoke System Generator

Read SGSCRIPT Process object code Produce load module file or load tape



7.1.6 Print Reports

A variety of reports, useful in various stages of software development and maintenance, is available. The reports are produced from data ase information which was explicitly or implicitly collected in the data base by the SPF. Representative information is: source code, software histories, software dependencies (nesting, externals), code statistics, translators used, etc.



7.2 Software Testing Procedures

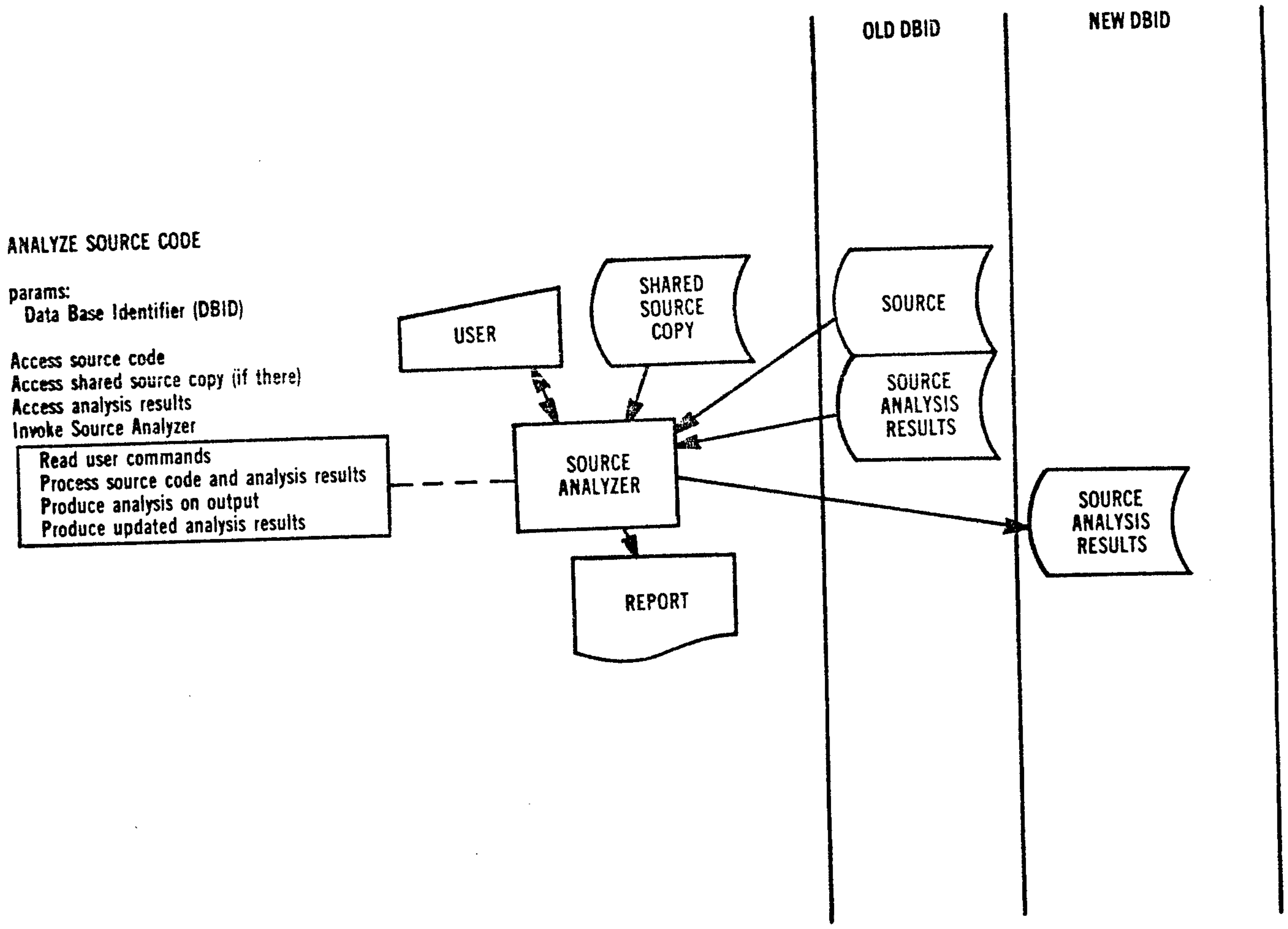
Testing is oriented toward software evaluation. The testers must be able to identify, develop and execute tests. They must also be able to evaluate the results and effectiveness of the tests.

		DB CONTENTS	
PROCEDURES	TOOLS	USED	PRODUCED
Analyze Source Code	Source Analyzer	Source Code Analysis Results	Analysis Results
Develop Tests	Test Case Generator Text Editor	Analysis Results Test Scripts Test Histories	Test Schema Data Test Scripts
Instrument Code	Source Instrumentor Translator System Generator	Source Code Source Instrumentor Script System Generator Script	Inst Source Inst Object Inst Load Module
Execute Tests	Computer Emulator	Test Script (Inst) Load Module	(Inst) Test Results Test Histories
Debug Tests	Computer Emulator	Source Code Test Scripts Load Modules	
Regression Test	Computer Emulator Test Results Comparator	Test Results Test Scripts Load Modules	Test Results Test Histories
Analyze Test Results	Test Results Analyzer	Inst Test Results	

Figure 6 - Summary of Software Testing Procedures

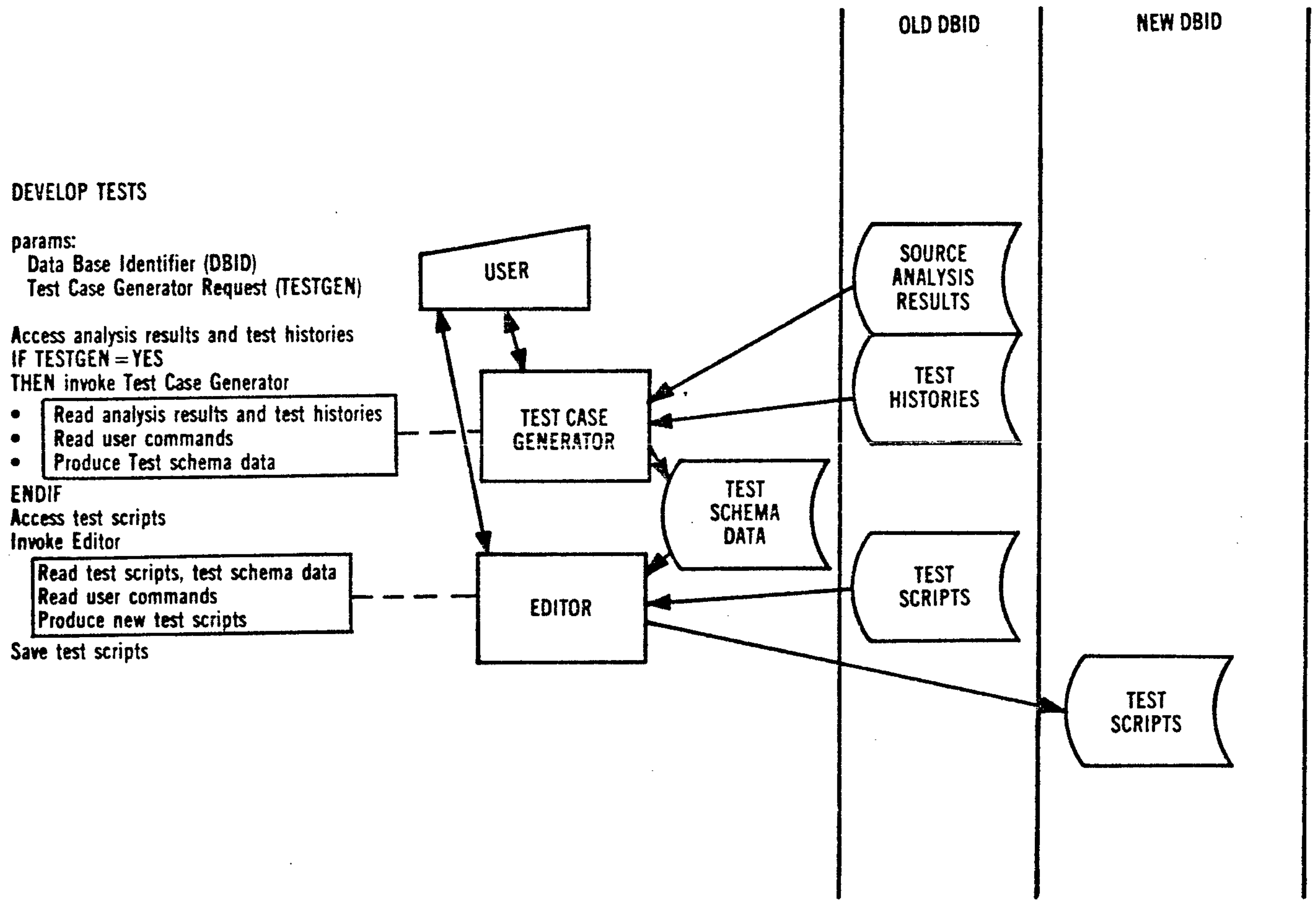
7.2.1 Analyze Source Code

A static analyzer examines source code and previous analyses to enforce standards, check for coding errors and perform quantitative analyses. It also provides information for test case development (e.g., value ranges of variables).



7.2.2 Develop Tests

A test consists of sequences of directives for a target computer emulator to specify the environment in which a load module will be executed as well as to specify the execution data to be sampled. Tests are saved as scripts and maintained on the data base just as system generator scripts are. Examples of environment specifications are memory and register initialization, and the values and timing of input data. Execution data include values in registers, memory, program counter, emulated clock timings and output data. Additional directives may identify those areas of test results which are important for regression test comparisons.



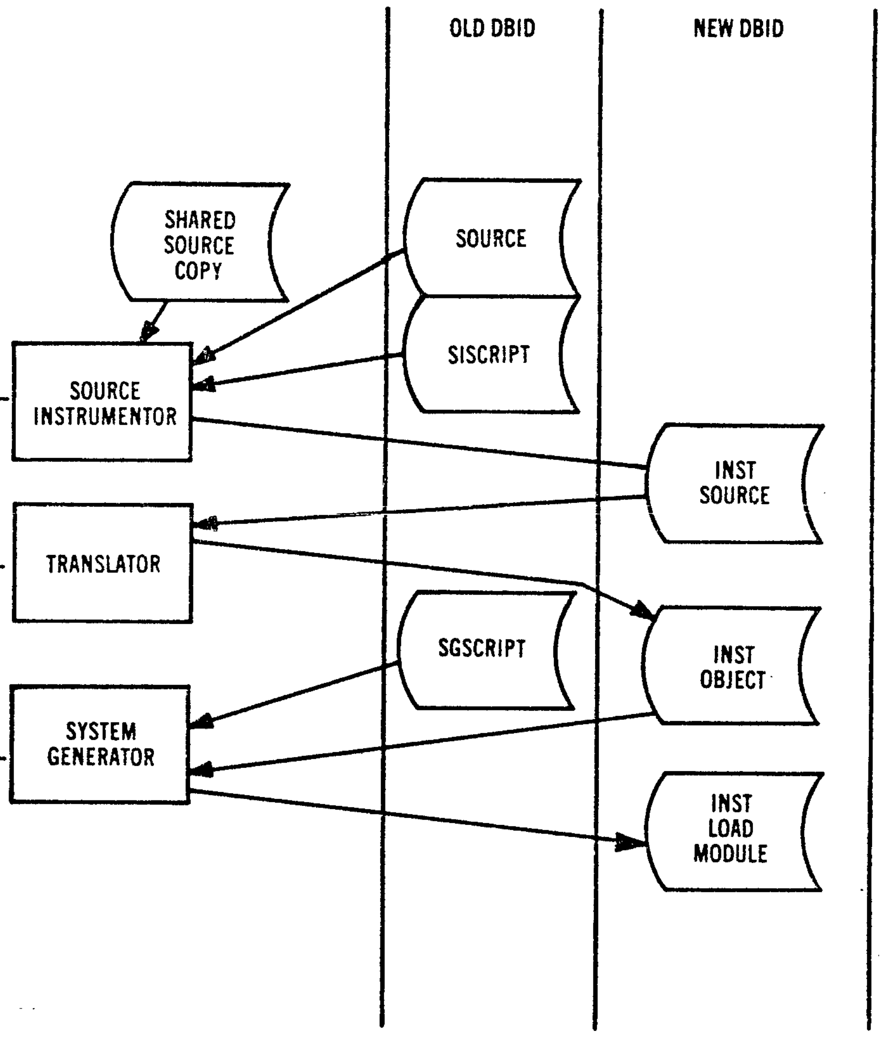
7.2.3 Instrument Code

Instrumentation of code provides a significant improvement in testing and evaluating test effectiveness. A source code instrumenter identifies the optimum locations for inserting "software probes" (calls to information-collecting programs) then instruments the source code by inserting the probes, and adding the programs. The source is automatically translated to create instrumented object code from which an instrumented load module is generated. All are saved on the data base.

```
INSTRUMENT CODE

params:
  Data Base Identifier (DBID)
  Source Code Translator (TRANSLATOR)
  Source Instrumenter Script (SISCRIPT)
  System Generator Script (SGSCRIPT)

Access source code and SISCRIPT
Access shared source copy (if there)
Invoke Source Instrumenter
  | Read SISCRIPT and source code |
  | Produce instrumented source   |
IF fatal errors
THEN raise abort flag
ELSE save instrumented source
•  Invoke TRANSLATOR
•   | Read instrumented source   |
•   | Produce instrumented object |
•  IF fatal errors
•  THEN raise abort flag
•  ELSE save instrumented object
•  •  Access SGSCRIPT
•  •  Invoke System Generator
•  •   | Read SGSCRIPT and instrumented object |
•  •   | Produce instrumented load module file |
•  •  IF fatal errors
•  •  THEN raise abort flag
•  •  ELSE save load module file
•  •  ENDIF
•  ENDIF
ENDIF
```



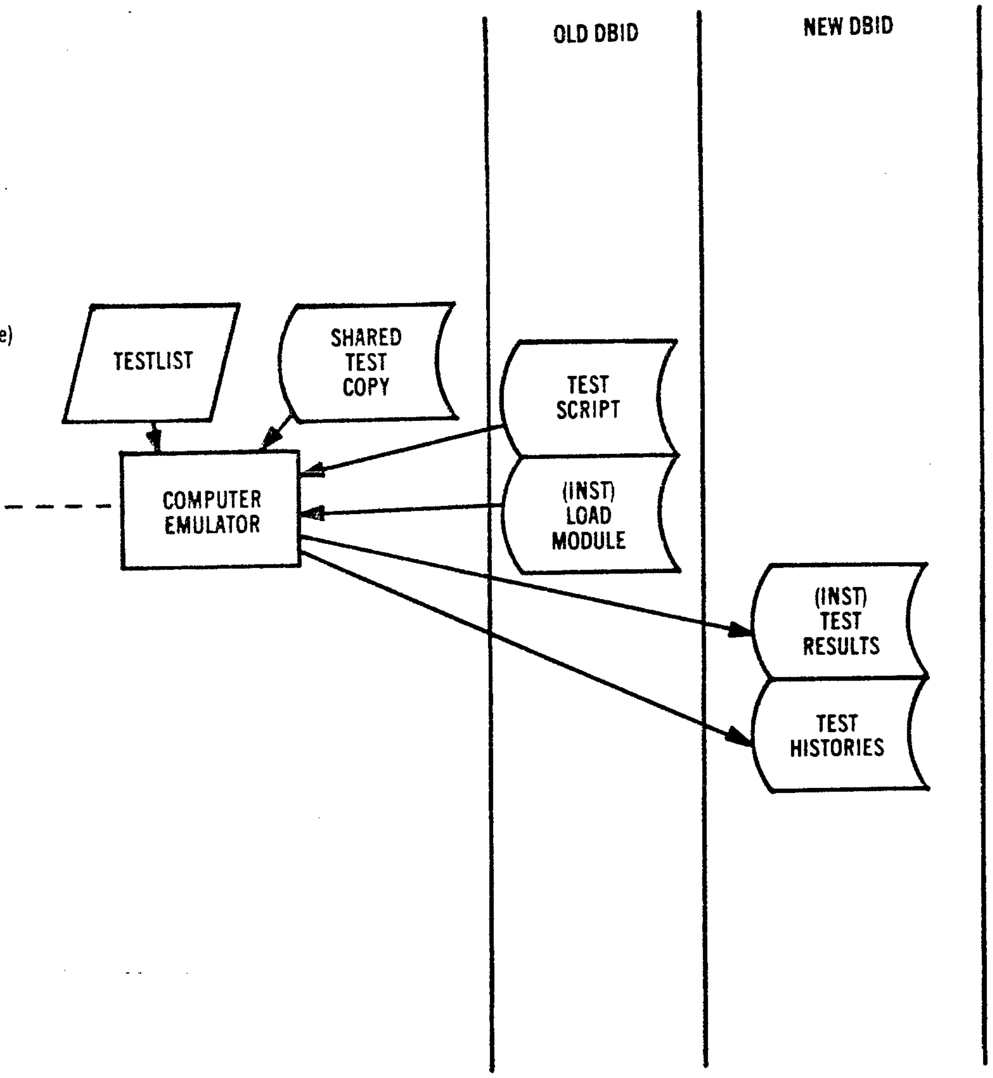
7.2.4 Execute Tests

Test execution uses a computer emulator program to evaluate development progress by executing load module code as the target hardware would; that is, it will go through the same internal states. But the fact that all emulated memory and registers are fully accessible at any point in the execution provides detailed visibility into the software execution. The existence of source code pointers in the object allows source-level direction and evaluation. Further, the use of instrumented load modules provides test effectiveness information.

```
EXECUTE TESTS

params:
  Data Base Identifier (DBID)
  List of Tests To Be Run (TESTLIST)

Access TESTLIST
Access shared test script copy (if there)
DOWHILE tests remain to be done
•  Access test script and load module
•    associated with test
•  Invoke Computer Emulator
•  | Load load module into memory |
•  | Process test script          |
•  | Produce test results         |
•  | Produce test histories       |
ENDDO
```



7.2.5 Debug Tests

To isolate problems in software or in test scripts, the SPF provides interactive control of the computer emulator. The computer emulator gives source code traces of instructions executed to aid the user in debugging source code.

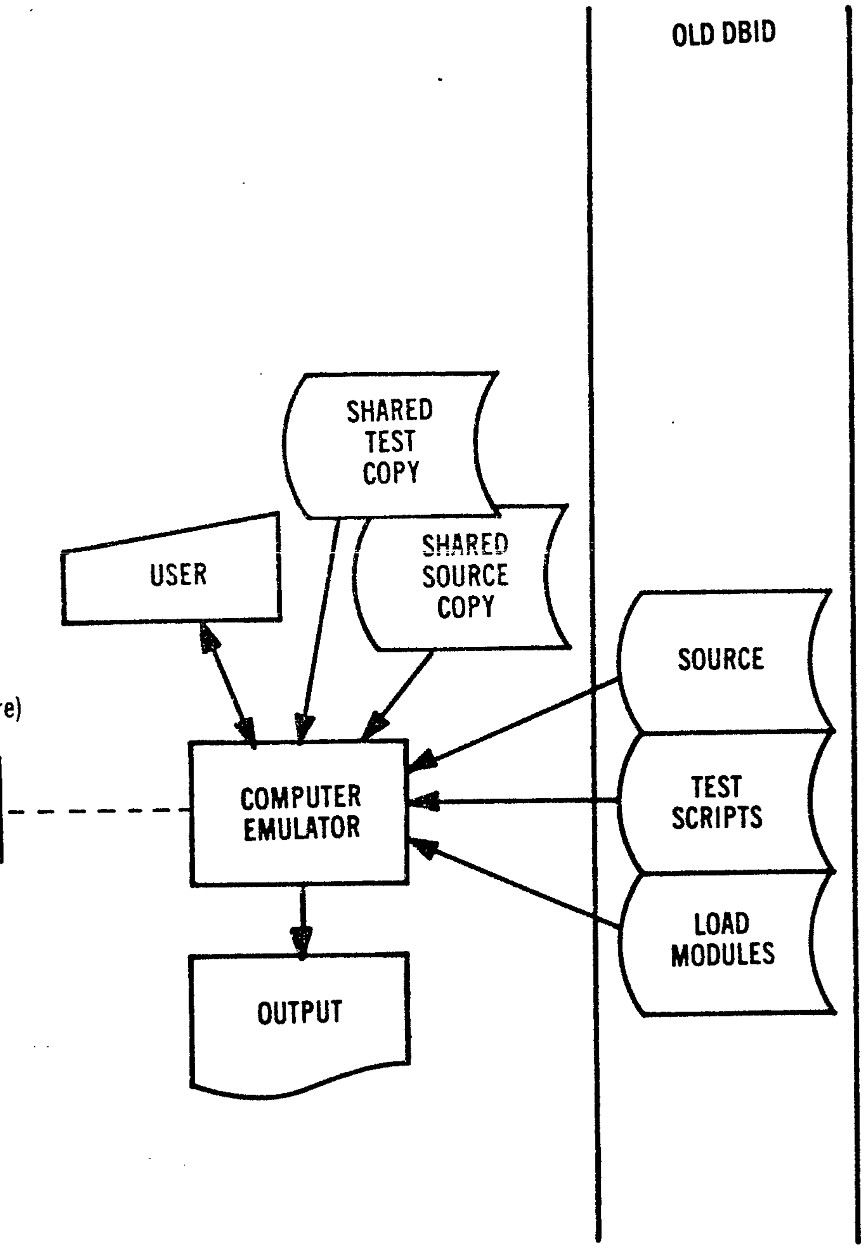
This procedure is for interactive use only; no test results are saved on the data base.

DEBUG TESTS

params:
  Data Base Identifier (DBID)

Access tests, source, loadmodules
Access shared test and source copies (if there)
Invoke Computer Emulator

Perform user commands (eg, load loadmodules, execute scripts) Provide source code traces



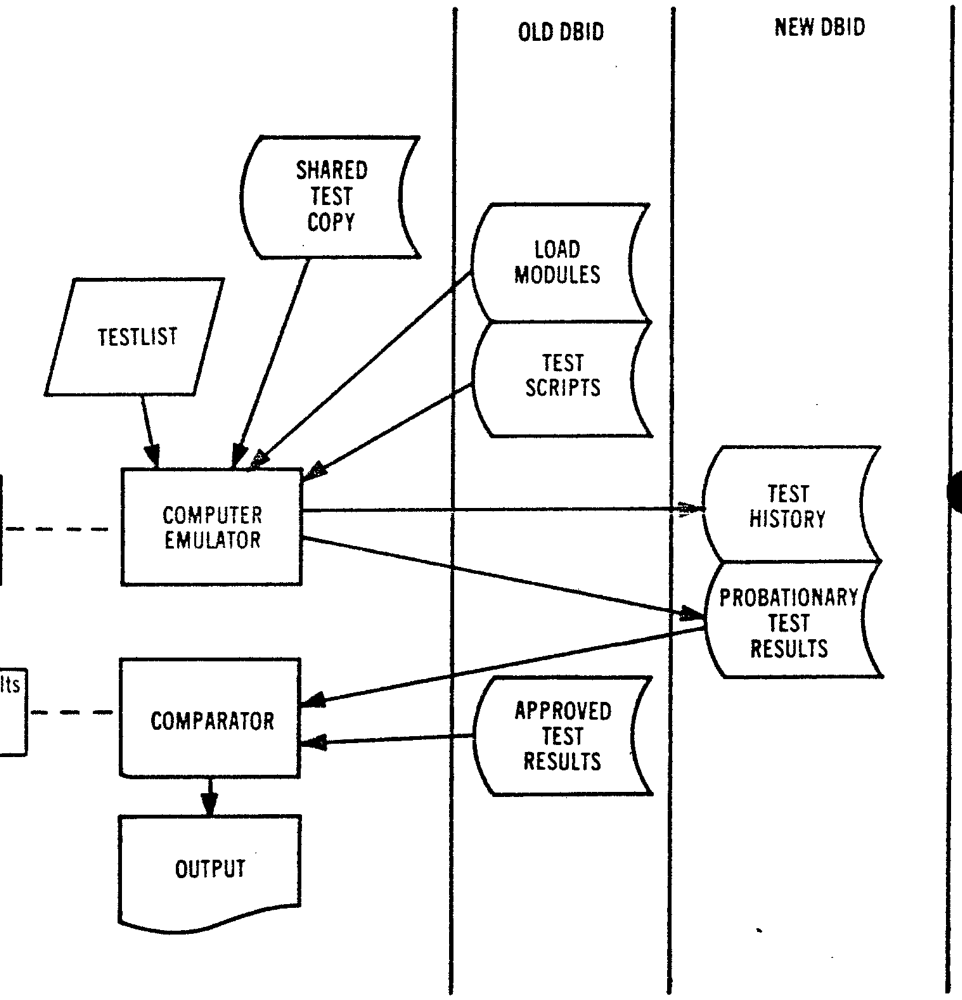
7.2.6 Regression Test

Tests are automatically or explicitly rerun when changes are made to determine whether any previously established capabilities have been compromised (i.e., have regressed). The sections of the new test results which are designated as "important" are compared with the same parts of the previous results. The new results are saved and the tester informed of any changes that occurred in important areas.

```
REGRESSION TEST

params:
  Data Base Identifier (DBID)
  List of Tests To Be Run (TESTLIST)

Establish TESTLIST of tests to be run
DOWHILE there are still tests to be run
•  Access next test and load module
•  Access shared test copy (if there)
•  Invoke Computer Emulator
•    Load loadmodule into emulated memory
•    Execute test script
•    Produce test results
•    Produce test history
•  Save test results as "probationary"
•  Access "approved" results
•  Invoke Comparator
•    Compare approved and probationary results
•    Identify important differences
•    Write Comparator findings to output
ENDDO
```



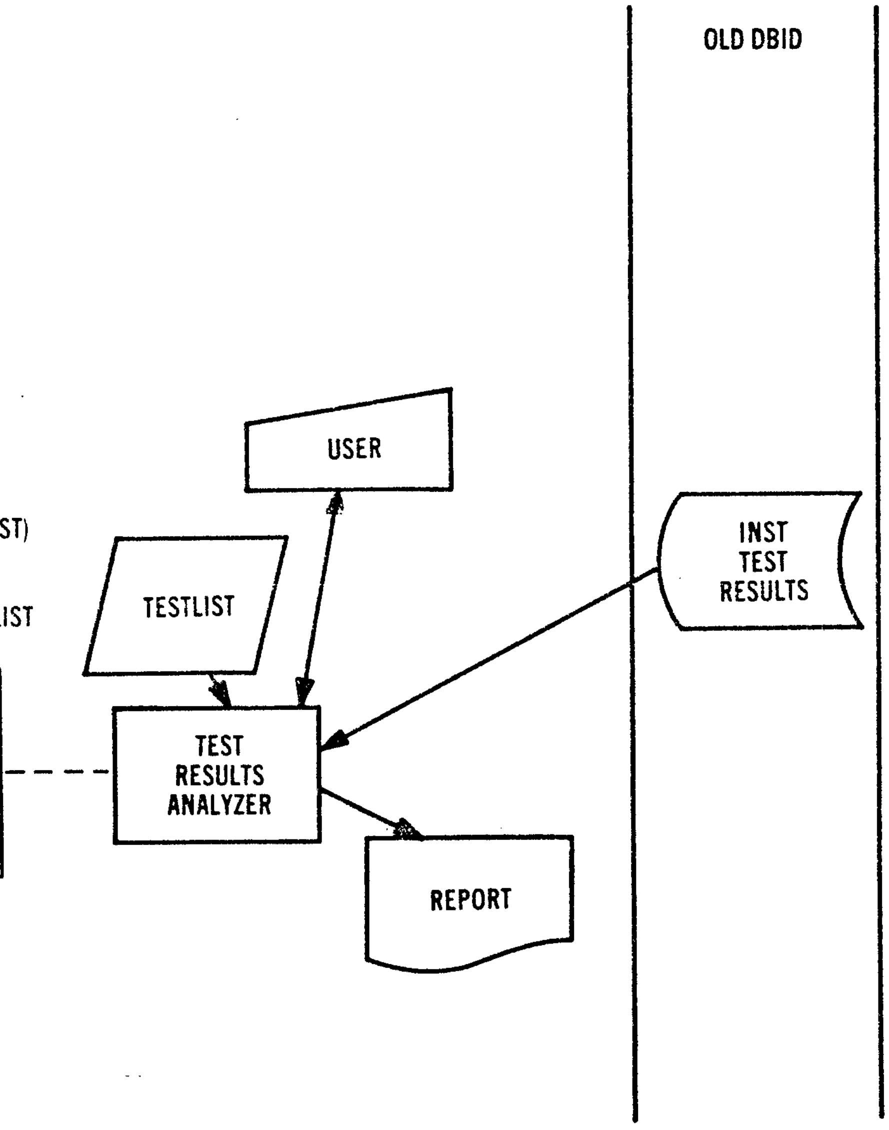
7.2.7 Analyze Test Results

A test results analyzer examines the results of testing instrumented object code to identify code paths highly used or not used at all, major time-usage areas and time allocation violations, actual ranges of variables and range assertion violations.

```
ANALYZE TEST RESULTS

params:
  Data Base Identifier (DBID)
  List of Tests To Be Run (TESTLIST)

Access instrumented test results
DOWHILE test name left in TESTLIST
•  Invoke Test Results Analyzer
•    Read test results
•    Read user commands
•    Produce information about:
•      path utilization
•      data ranges encountered
•      assertion violations
•      time usage
ENDDO
```



8. COMMON PROCEDURE GROUPS

8.1 User Assistance Procedures

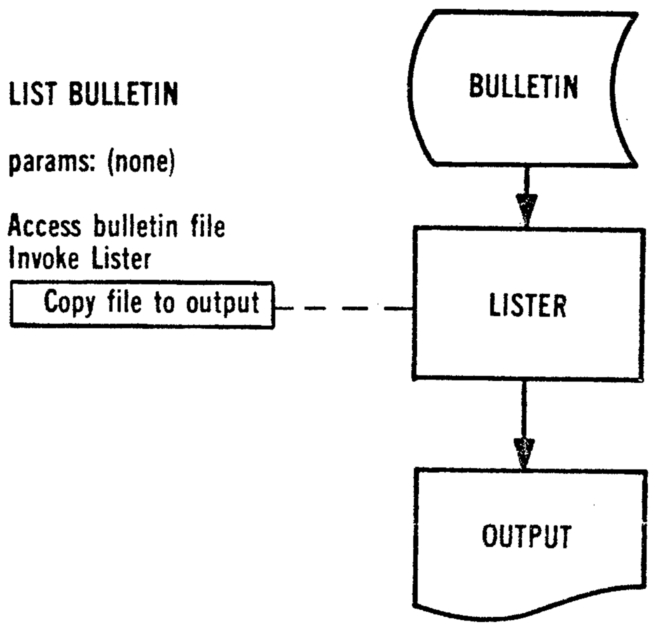
The SPF must have adequate documentation about available features and usage, and about current and future events affecting SPF usage. They must also be able to communicate with SPF personnel for additional aid. User assistance procedures are invoked by immediate commands.

PROCEDURES	TOOLS
List Bulletin	Lister
List News	Lister
List Help	Lister
List User Manual	Lister
Contact SPF Personnel	Electronic Mail System

Figure 7 - Summary of User Assistance Procedures

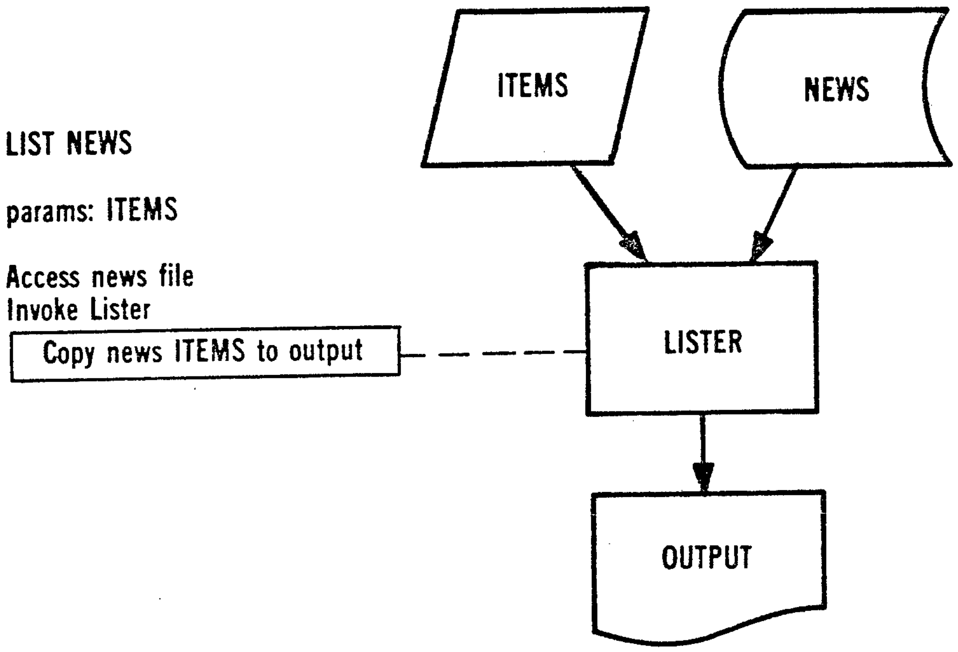
8.1.1 List Bulletin

This procedure produces a brief note concerning current or future events for SPF users. A bulletin is generally a notice that refers the user to the new user's manual or news volumes for details. When a user runs a batch job, this procedure is automatically invoked, causing the message to be printed on the first page of the output listing.



8.1.2 List News

The news is an SPF-maintained library of timely narratives which contain the details omitted from the bulletin. News items are available individually or in groups.



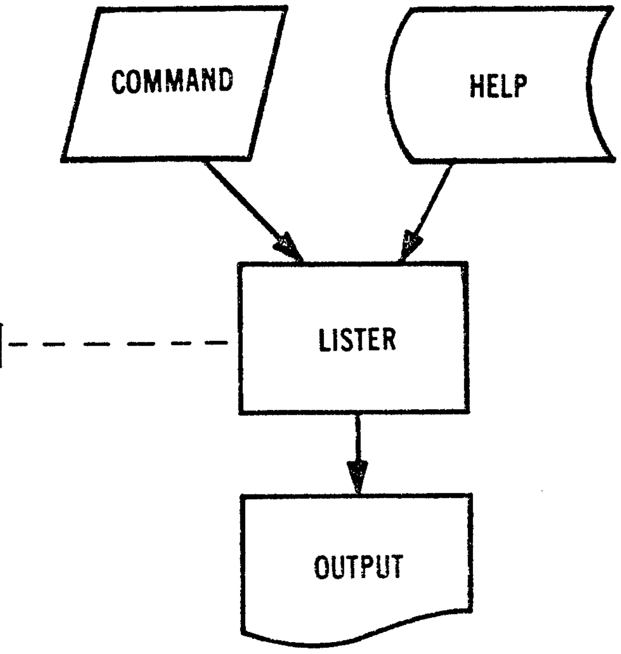
8.1.3 List Help

"Help" is stable information about SPF procedures and their commands, providing such information as data formats required, notable features and examples of use. Additional help is typically available at several levels of SPF usage, both inside and outside of tool invocations.

LIST HELP

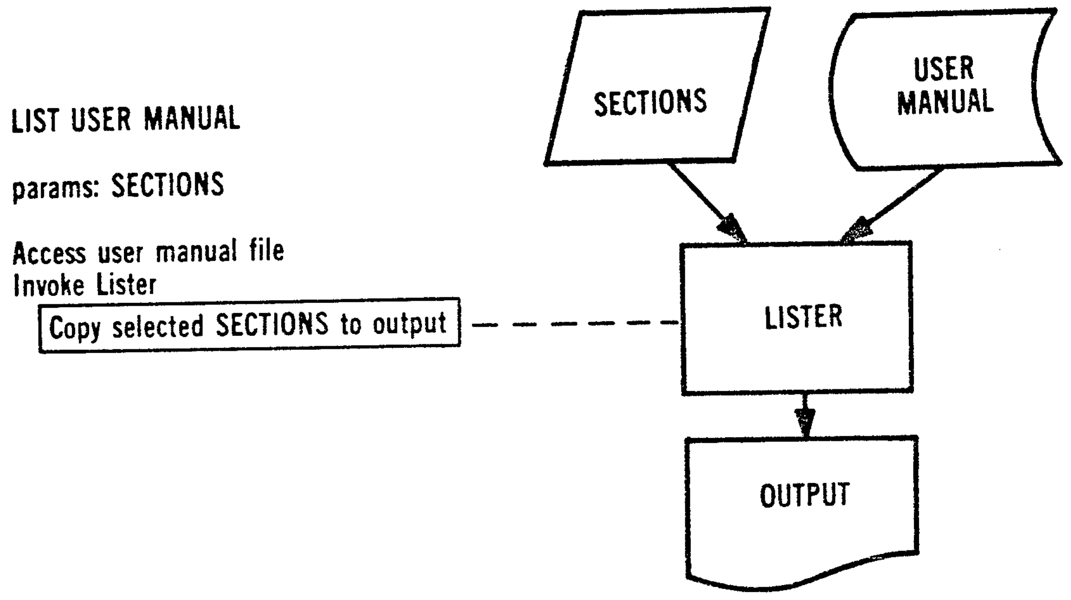
params: **COMMAND**

Access help file
Invoke Lister
    Copy COMMAND information to output



8.1.4 List User Manual

A new copy of the current User Manual is available for printing by the SPF user. Individual sections, or the entire new manual may be printed.



8.1.5 Contact SPF Personnel

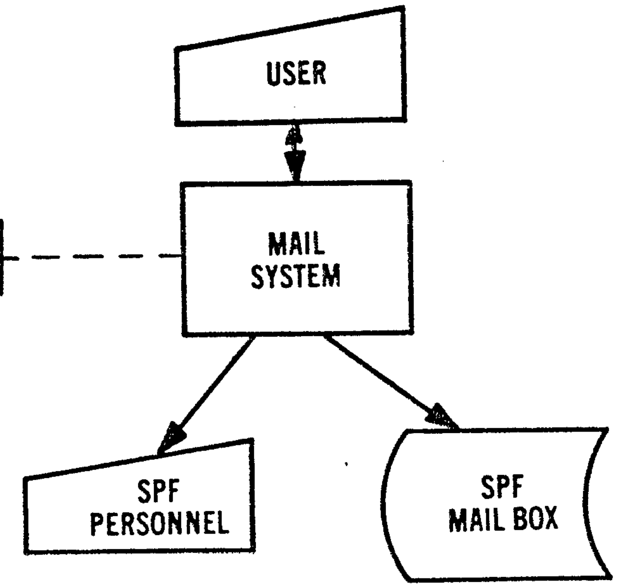
This procedure enables users to mail a message to SPF maintenance personnel or permits interactive communication.

CONTACT SPF PERSONNEL

params: (none)

Invoke Electronic Mail System

Send message to SPF personnel or enable two-way communication



8.2 Software Management Procedures

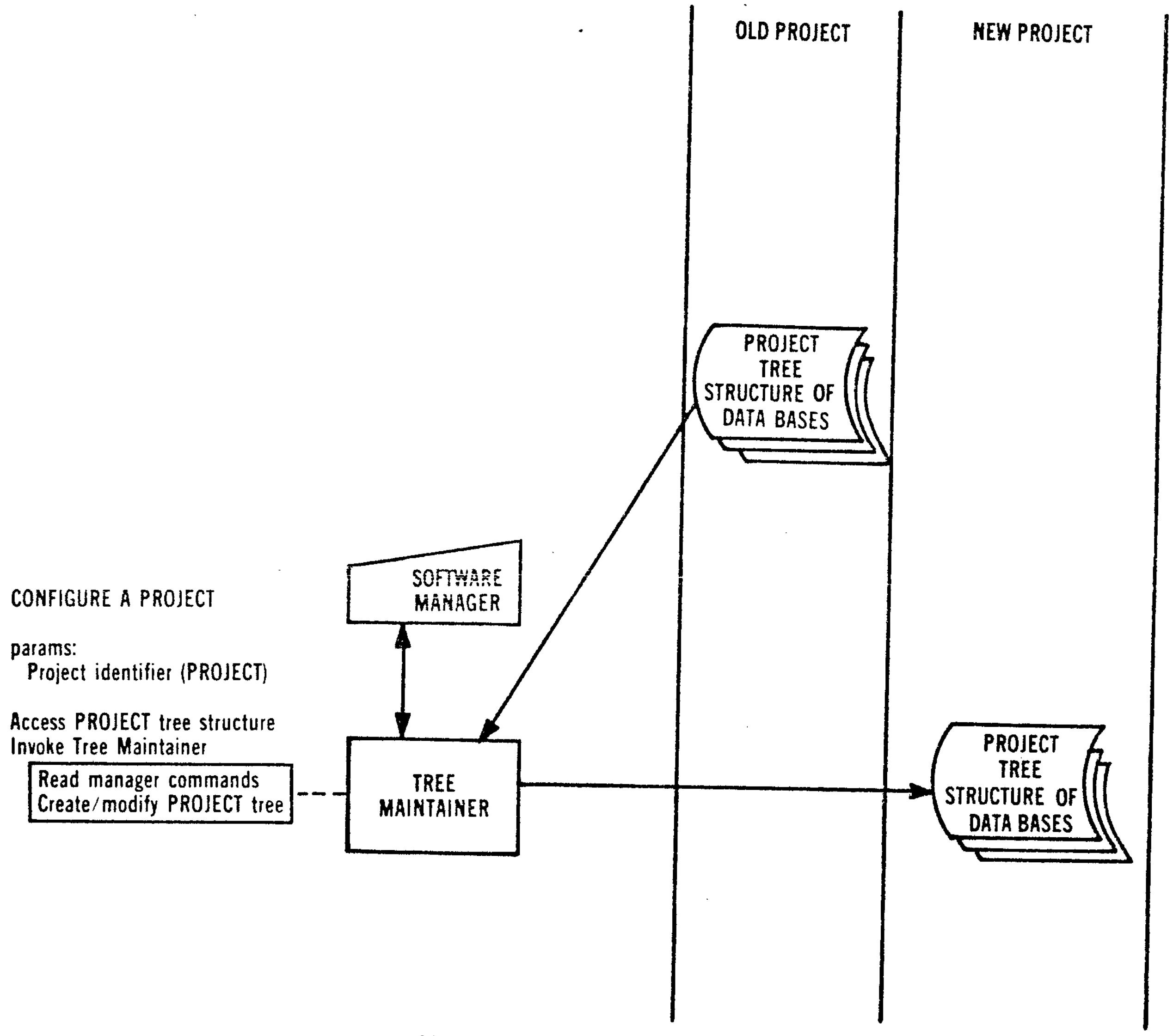
The SPF must also be an effective management tool, providing visibility and control of the software development and maintenance process. The manager must be able to control and get reports on his/her subordinates, software data bases, software products, and documentation. The reports must identify development progress against projections, provide SPF usage information, and provide for software configuration management.

		DB CONTENTS	
PROCEDURES	TOOLS	USED	PRODUCED
Configure a Project	Tree Maintainer	Project Tree Structure	Project Tree Structure
Control Access	Keys Maintainer	Keys Project Tree Structure	Keys
Print Progress Reports	Report Generator	DB Contents	
Identify SW Configuration	Project Data Extractor	Project File Contents	
Release SW		DB	DB
Track STRs/SCPs/SEPs	Tracker	STR/SCP/SEP Log	STR/SCP/SEP Log
Configuration Status Accounting	Project Data Extractor	Project Files STR/SCP/SEP Log	

Figure 8 - Summary of Software Management Procedures

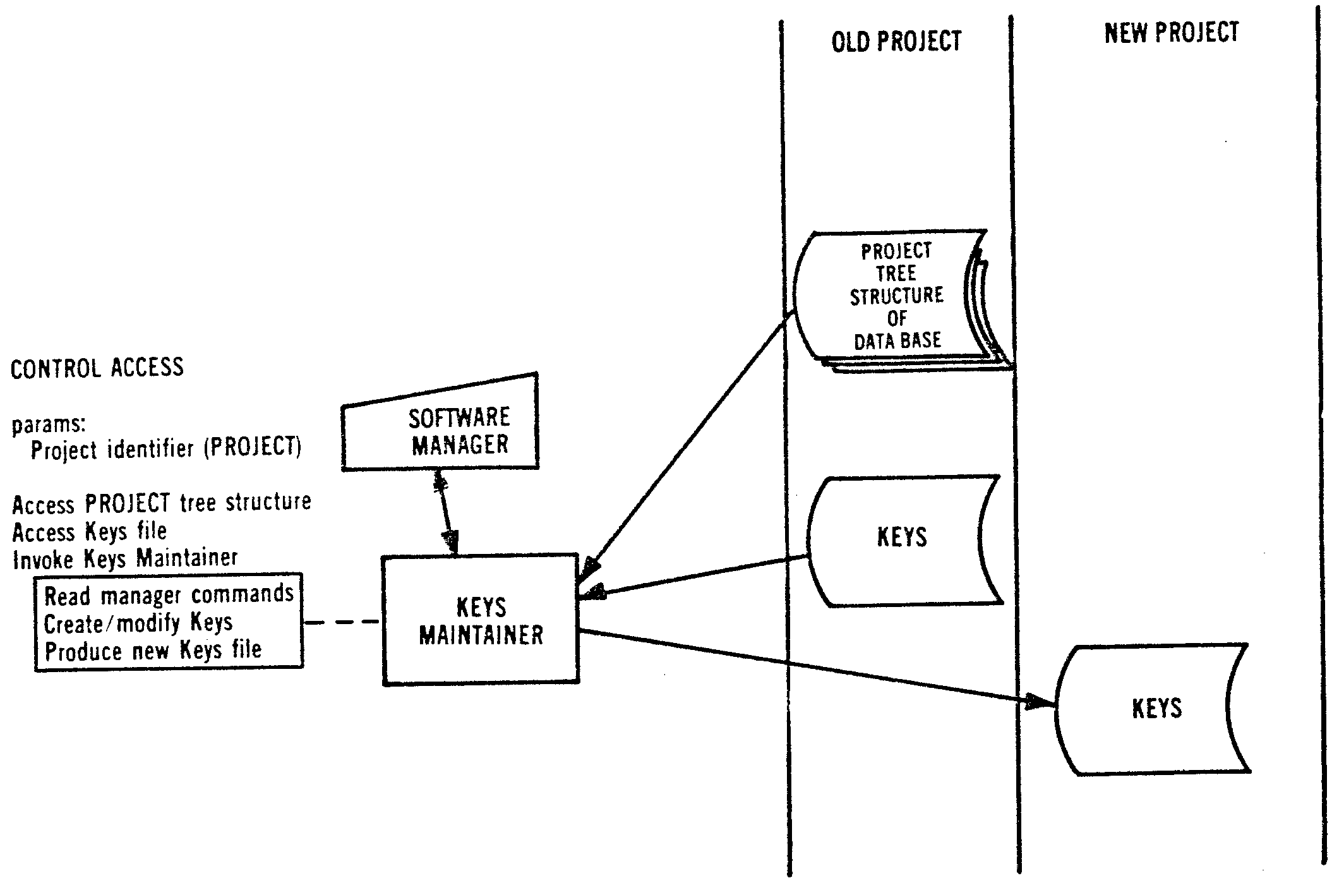
8.2.1 Configure a Project

The software manager defines a project tree structure that reflects the hierarchy of the work groups and software products comprising his/her project, as shown in Figure 1. The leaves of the tree are the data bases.


OLD PROJECT
NEW PROJECT
PROJECT TREE STRUCTURE OF DATA BASES
SOFTWARE MANAGER
CONFIGURE A PROJECT
params:
Project identifier (PROJECT)
Access PROJECT tree structure
Invoke Tree Maintainer
Read manager commands
Create/modify PROJECT tree
TREE MAINTAINER
PROJECT TREE STRUCTURE OF DATA BASES

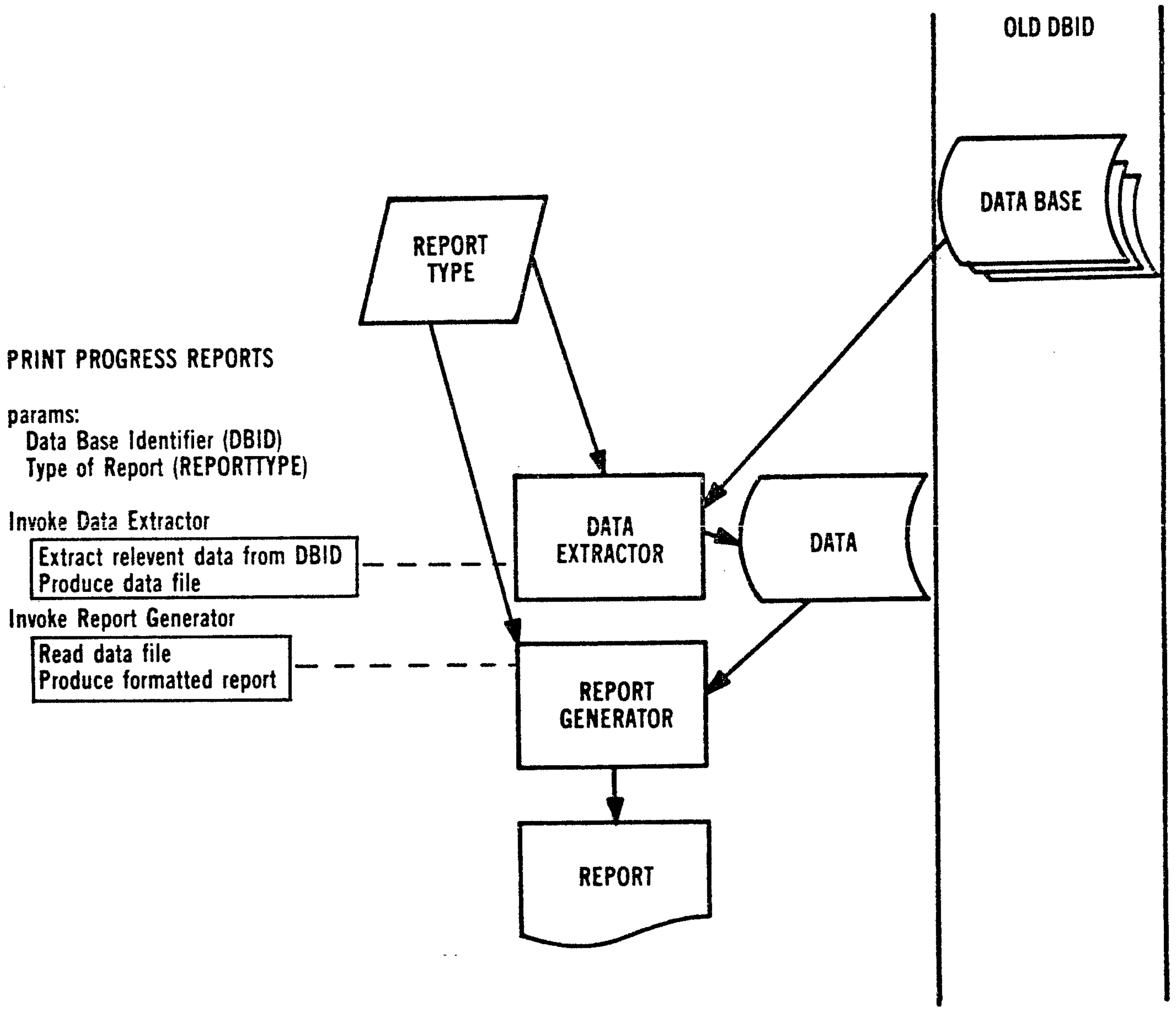
8.2.2 Control Access

The software manager controls creation and modification of the project tree structure. He/she can limit access to limbs or leaves of the tree to specified users as well as restricting the use of SPF procedures to specified users. Examples of procedure restrictions are source or test modification, sharing or copying from a data base, use of alternate translator versions and establishing or changing user access keys. The access keys are the means by which the manager controls the project.



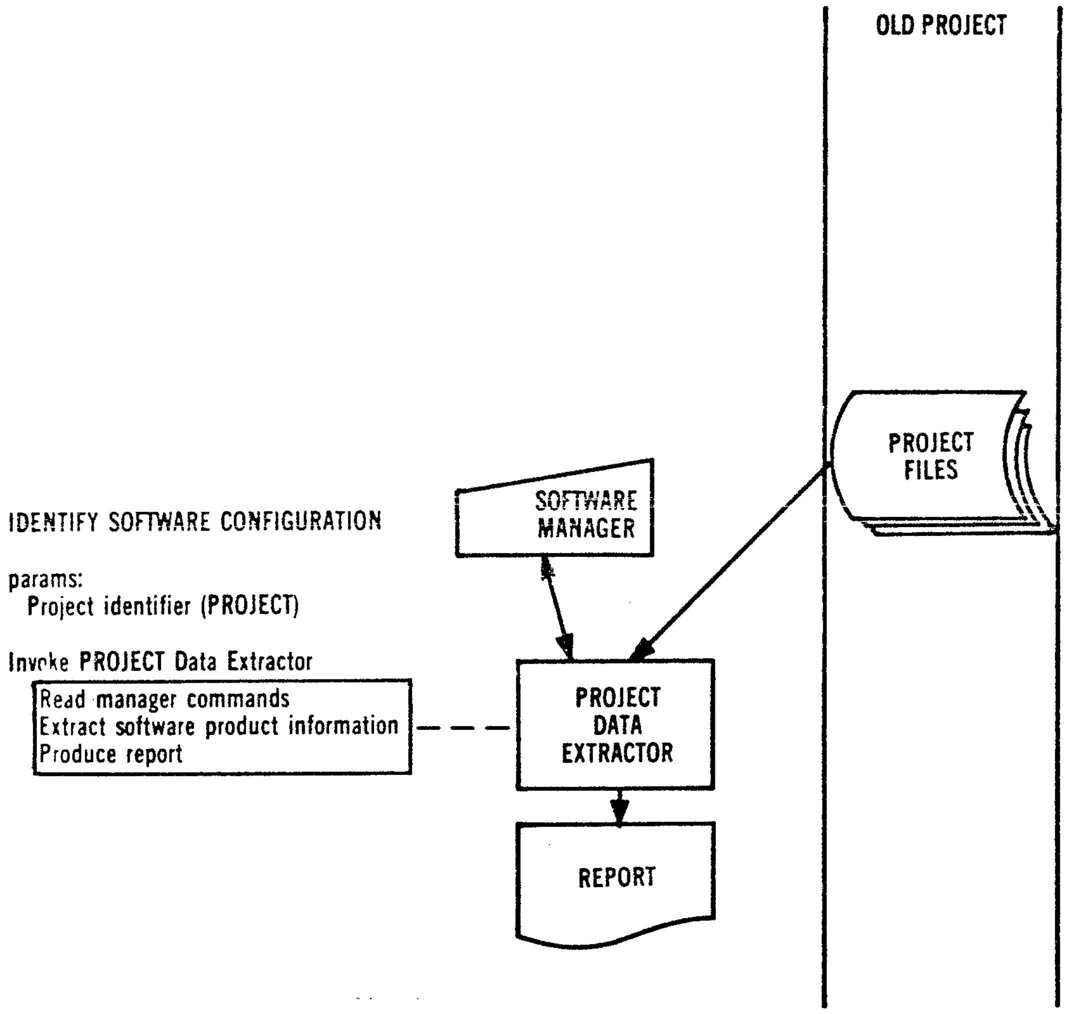
8.2.3 Print Progress Reports

A report generator produces a variety of reports on the software development process from information gathered by the DBMS. The reports include software development histories, testing histories, software metrics, SPF utilization, cost accounting, and job statistics.



8.2.4 Identify Software Configuration

Configuration reports are produced for specified software products identifying each of their component elements in the project tree. The report may be produced for that portion of the project for which the requesting manager has the access rights. For example, given the structure in figure 1, the MAD manager could not get a report for the entire AOP, but the Operational Software manager could. The reports include software histories, and software statistics (metrics).


OLD PROJECT
PROJECT FILES
SOFTWARE MANAGER
IDENTIFY SOFTWARE CONFIGURATION
params:
Project identifier (PROJECT)
Invoke PROJECT Data Extractor
Read manager commands
Extract software product information
Produce report
PROJECT DATA EXTRACTOR
REPORT

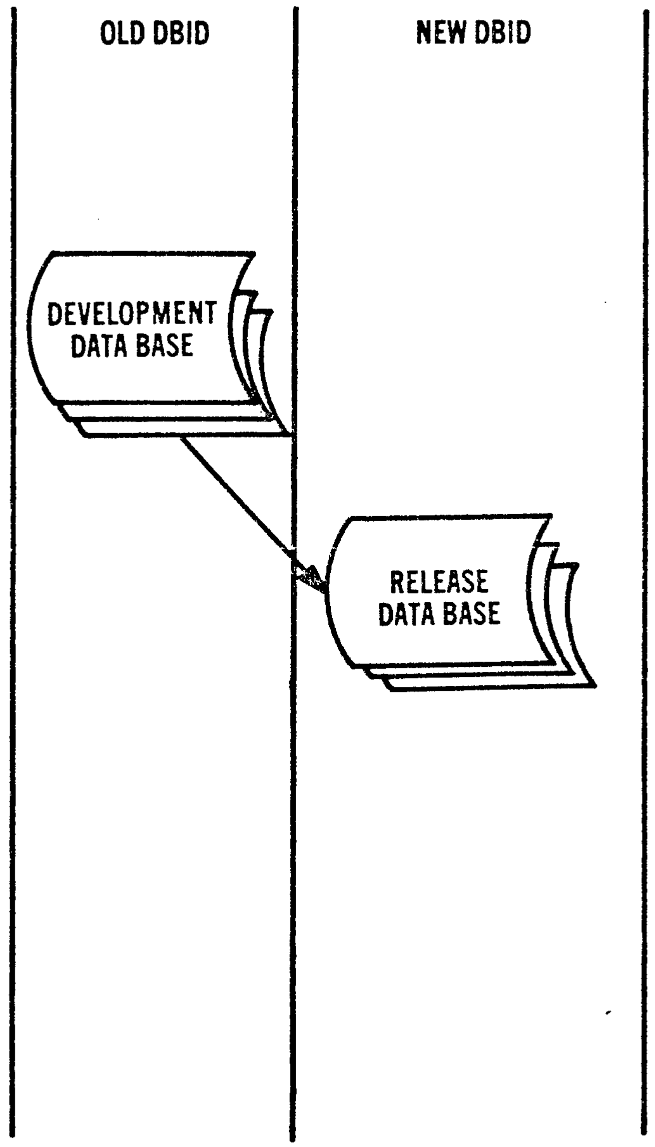
8.2.5 Release Software

This procedure transitions software from a development environment to a maintenance or quality assurance environment where configuration status accounting, access restrictions and access reporting may be performed in a more rigorous fashion. At this point the software elements are linked to an STR/SCP/SEP section of data base for control and/or reporting purposes.

RELEASE SOFTWARE

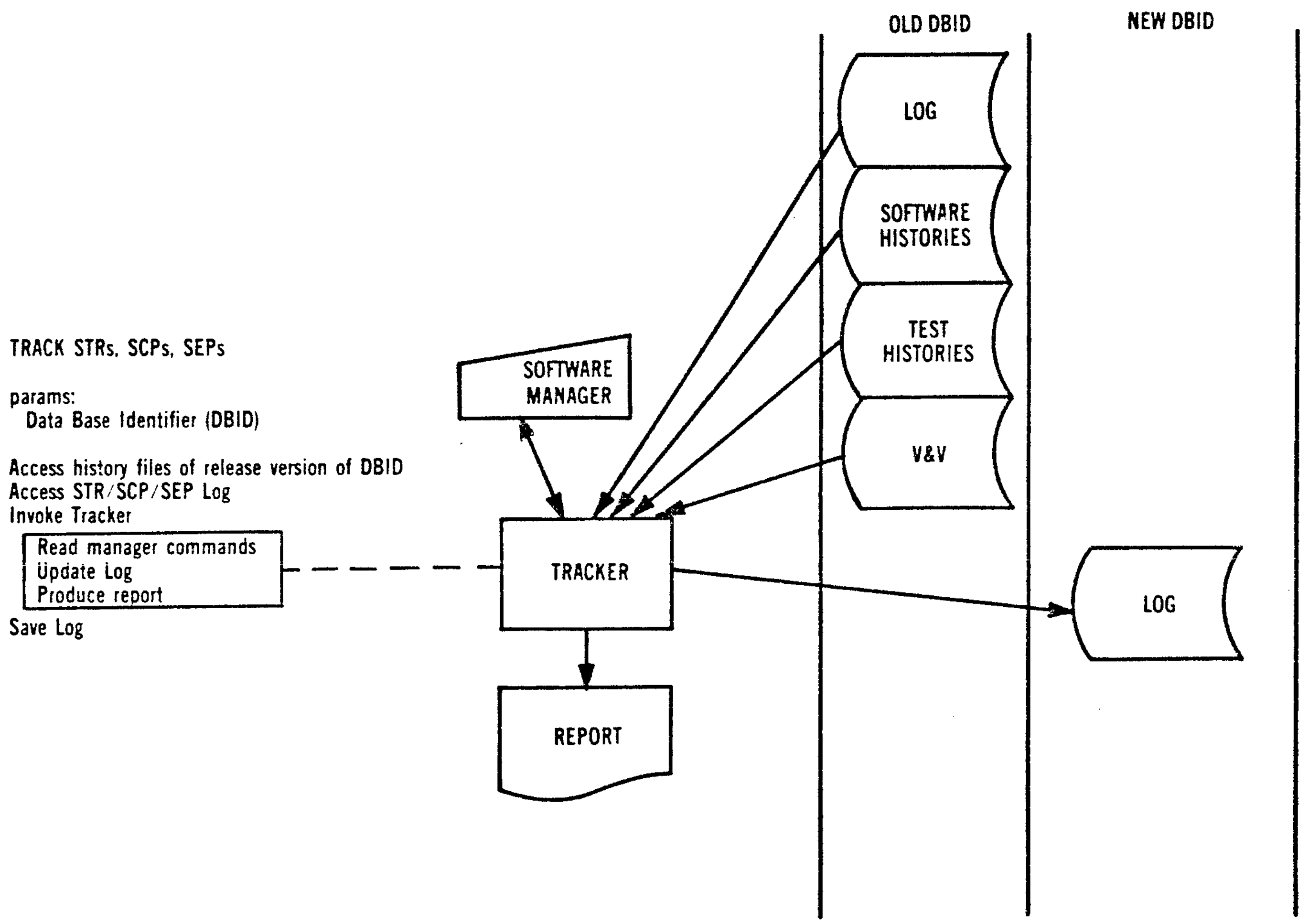
params:
  Data Base Identifier (DBID)

Copy development version of DBID
  to release version



8.2.6 Track STRs, SCPs, and SEPs

The SPF maintains a project log of STRs/SCPs/SEPs. This log is linked to released software for control and/or reporting purposes. The SPF will maintain all necessary information about the STR/s/SCPs/SEPs and will provide reports on status of all or selected items from the log as requested.



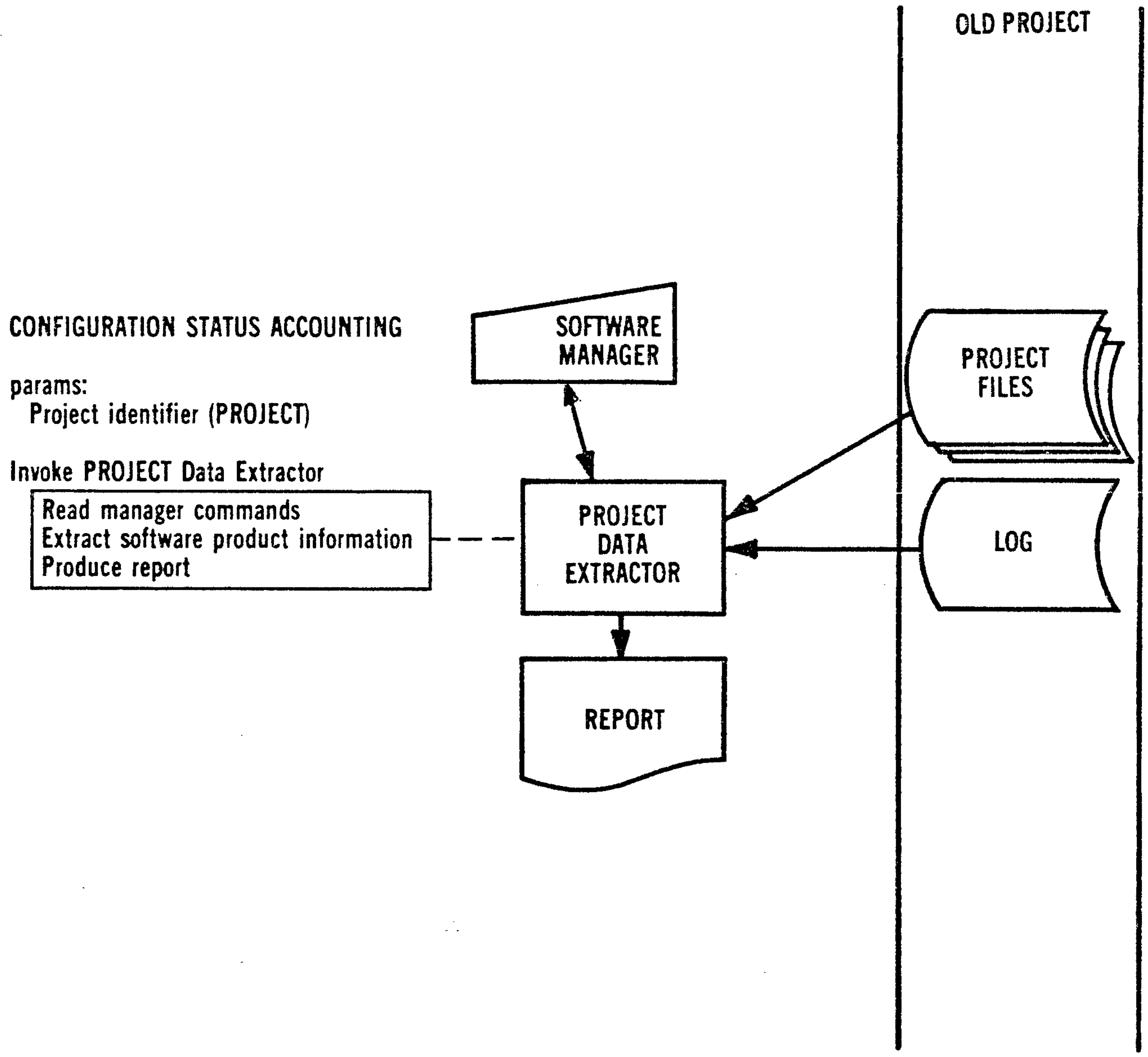
TRACK STRs, SCPs, SEPs

params:
  Data Base Identifier (DBID)

Access history files of release version of DBID
Access STR/SCP/SEP Log
Invoke Tracker
  Read manager commands
  Update Log
  Produce report
Save Log

8.2.7 Configuration Status Accounting

An audit report is produced showing the element hierarchy, part number, and version number for each element. An expanded report includes the date of all changes, the STR/SCP/SEP number that initiated the change, the name of the engineer who made the change, and the validation and verification dates for the element.


OLD PROJECT
CONFIGURATION STATUS ACCOUNTING
params:
Project identifier (PROJECT)
Invoke PROJECT Data Extractor
Read manager commands
Extract software product information
Produce report
SOFTWARE MANAGER
PROJECT FILES
PROJECT DATA EXTRACTOR
LOG
REPORT

8.3 Documentation Production Procedures

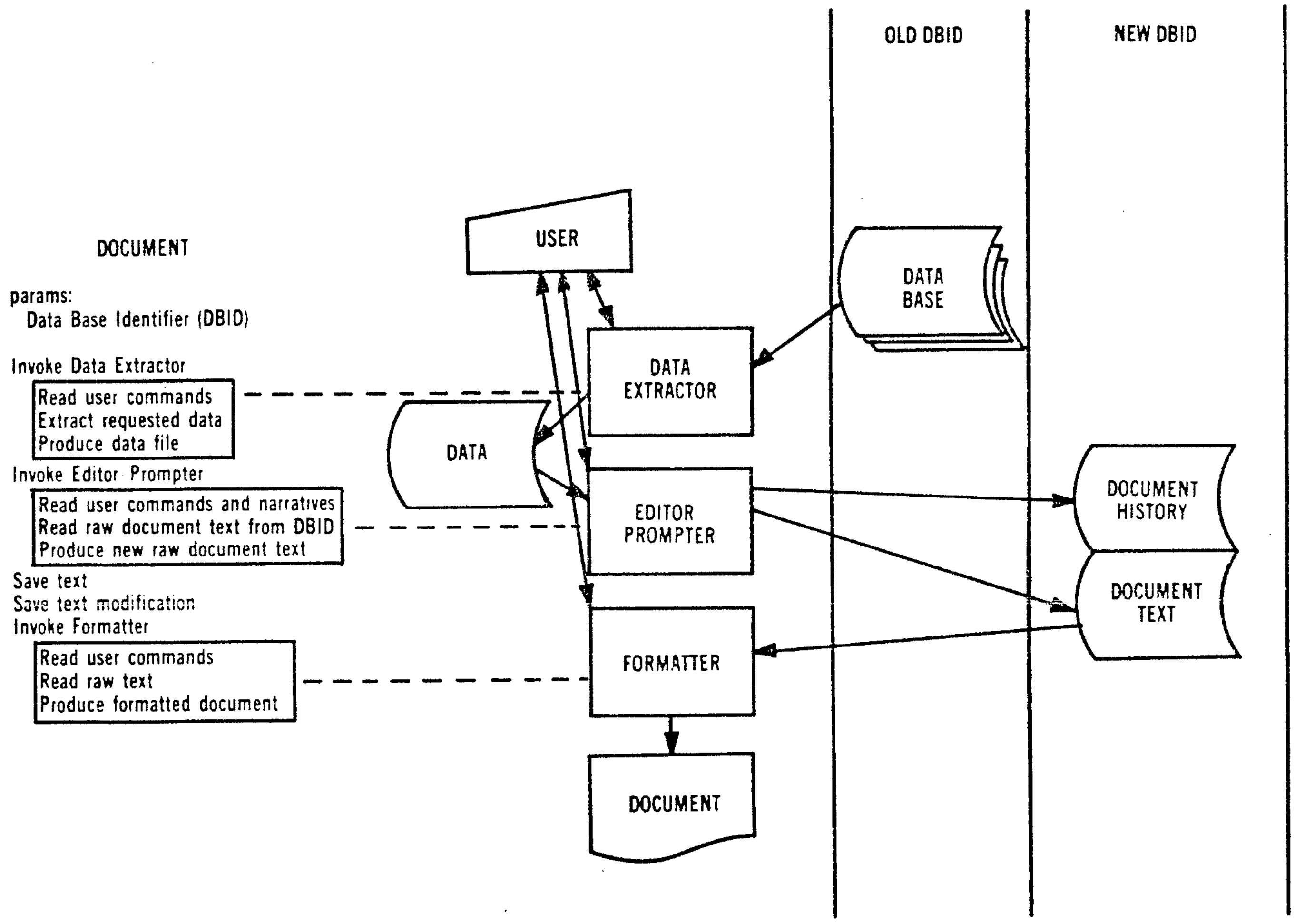
Descriptive documents can be produced semi-automatically from the information collected by the SPF from the software production activity.

		DB CONTENTS	
PROCEDURES	TOOLS	USED	PRODUCED
Document	Data Extractor Text Editor/Prompter Document Formatter	Document Text Document History	Document Text Document History

Figure 9 - Summary of Document Production Procedures

8.3.1 Document

The user specifies how data is to be selected and processed and provides narrative text through a text editor/prompter. A document formatter produces the final document.



8.4 Context Control Procedures

Context control provides the user with necessities and valuable conveniences not covered in the other common procedure groups (e.g., controlling the command queue, setting global values). By their nature, context control procedures are invoked by immediate commands.

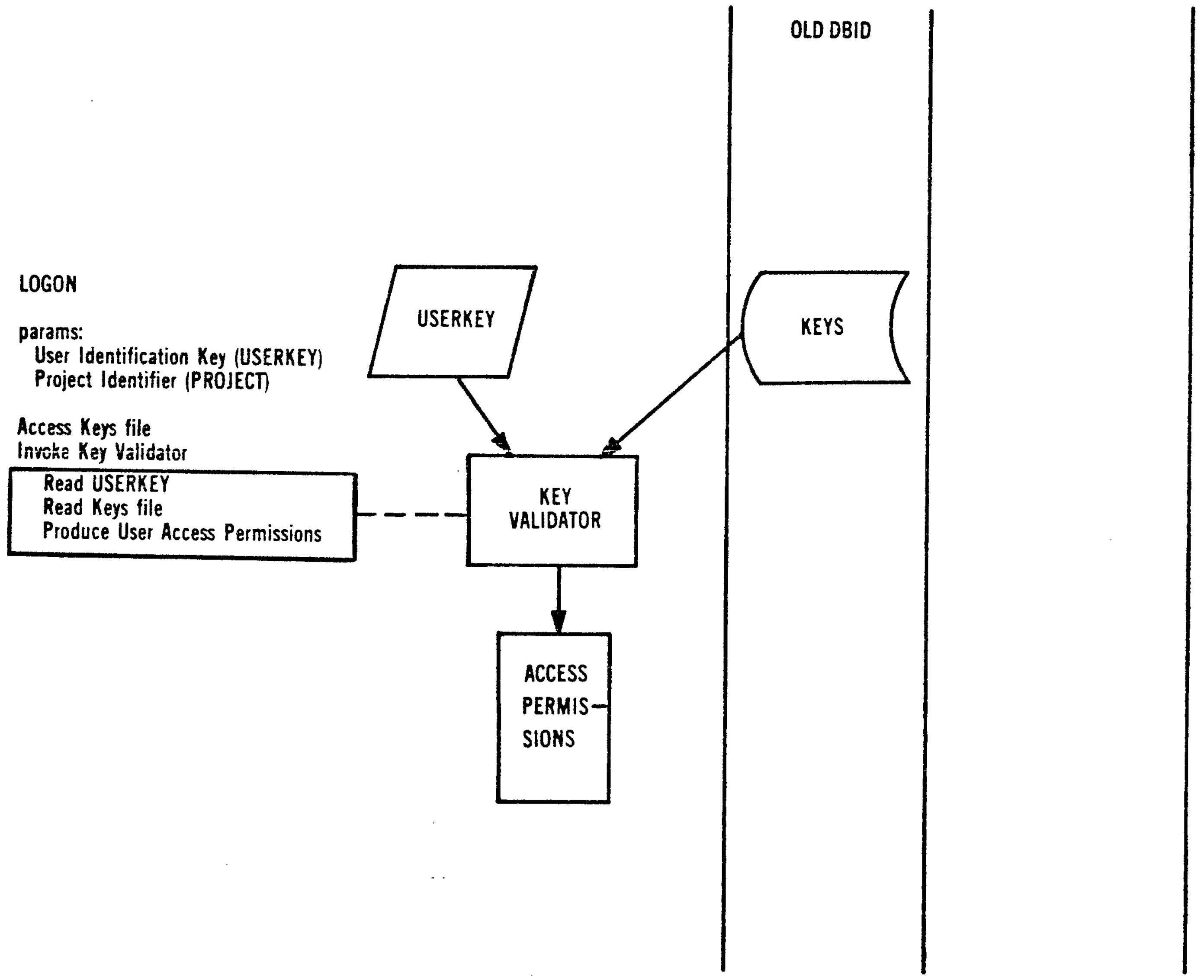
		DB CONTENTS	
PROCEDURES	TOOLS	USED	PRODUCED
Logon	Key Validator	Keys	
Logoff	Wrapup		System Utilization Record
List Global Parameters	Lister		
Set/Clear Global Parameters	Globals Maintainer		
List Command Queue	Lister		
Activate/Deactivate Command Queue	Command Queue Maintainer		
Execute Command Queue	Command Queue Executor		
Generate Scripts	Editor/ Prompter		Scripts
Examine Output	Text Scanner		
Save Output	DBMS		Output Listings
Print Output	Lister		

Figure 10 - Summary of Context Control Procedures

8.4.1 Logging Procedures

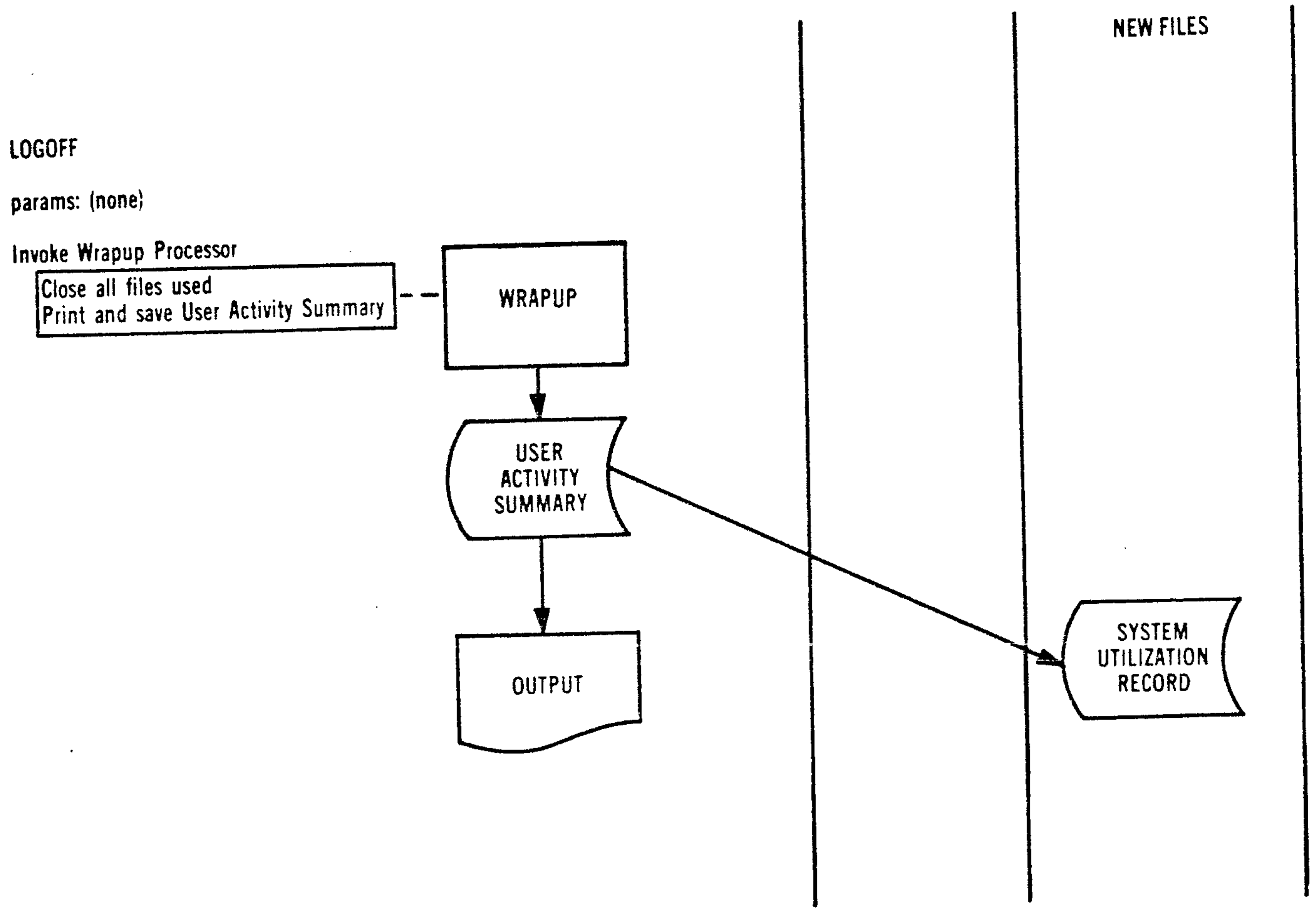
8.4.1.1 Logon

The user identifies him/herself and the project to be worked on. The list of access permissions assigned to the user is made available to subsequently executed procedures.



8.4.1.2 Logoff

The SPF terminates the session, ties up loose ends, establishes system usage statistics which are both printed and recorded for accounting purposes.



8.4.2 Global Parameter Handling Procedures

The SPF affords the user the one-time setting of default values for frequently used parameters.

8.4.2.1 Set Global Parameters

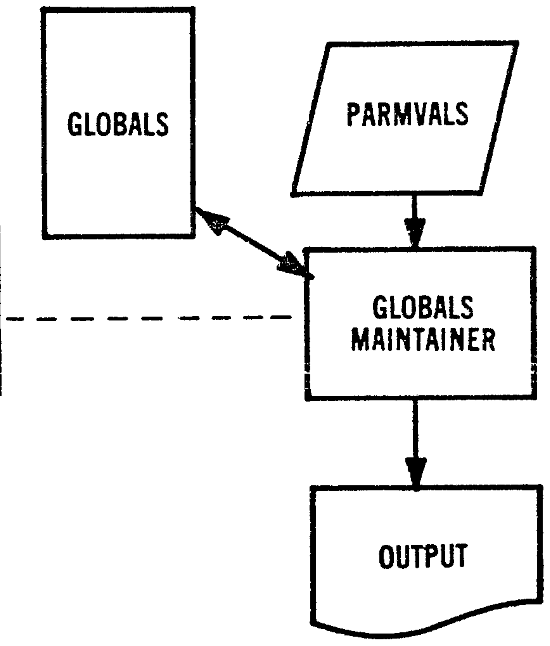
The user specifies parameter values to be applied as defaults for subsequent commands.

SET GLOBAL PARAMETERS

params:
  List of any param = value (PARMVALS)

Access Global Parameter List (Globals)
Invoke Globals Maintainer

- Read Globals
- Read input PARMVALS
- Validate parameters and values
- Add/change valid PARMVALS in Globals
- Notify user of invalid entries



8.4.2.2 Clear Global Parameters

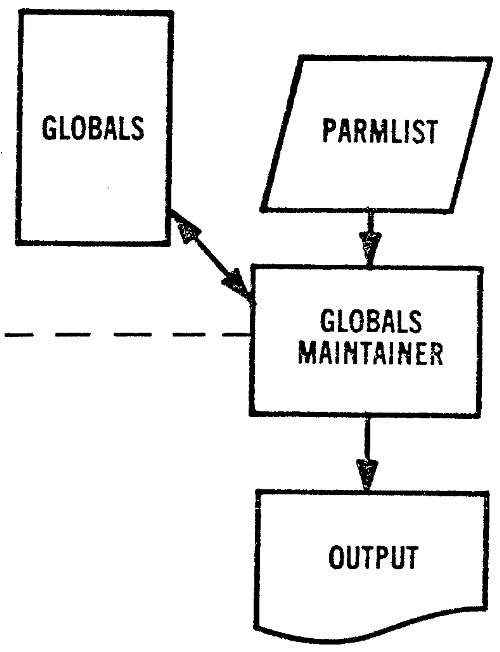
The user specifies parameter names to be reset to system default values.

CLEAR GLOBAL PARAMETERS

params:
  List of any param names (PARMLIST)

Access Global Parameter List (Globals)
Invoke Globals Maintainer

Read Globals
Read input PARMLIST
Remove valid PARMLIST names from Globals
Notify user of any invalid entries



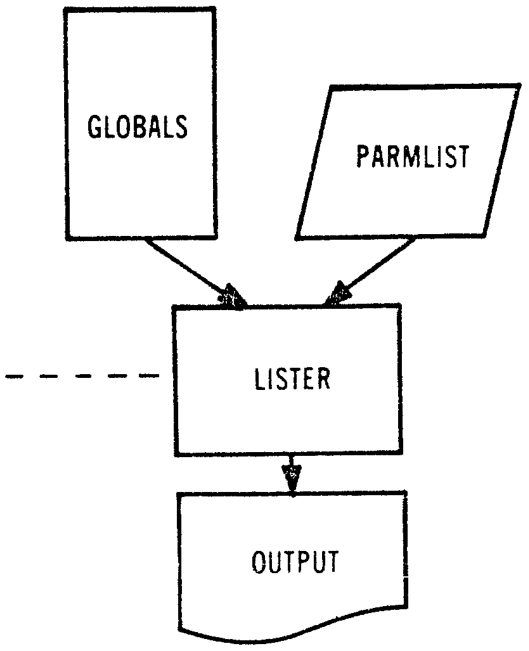
8.4.2.3 List Global Parameters

The SPF provides a listing of each globally set parameter and its current default value.

```
LIST GLOBAL PARAMETERS

params:
  List of any parameter names (PARMLIST)

Access Global Parameter List (Globals)
Invoke Lister
  Read Globals
  Read input PARMLIST
  Copy valid PARMLIST values to output
  Notify user of invalid entries
```



8.4.3 Command Queue Handling Procedures

The SPF allows the user to inspect the command queue contents, change the activation status of any entry and execute the activated commands.

8.4.3.1 Activate/Deactivate Command Queue

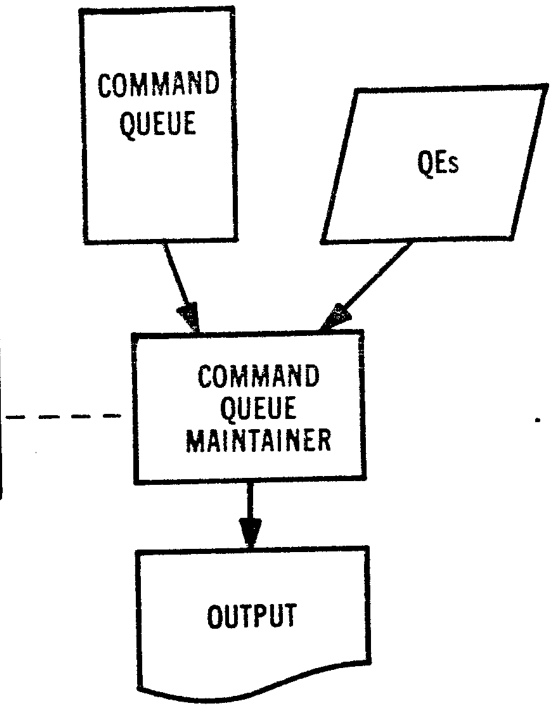
A command is normally activated when initially placed on the queue and deactivated when executed. This command changes the activation status of specified commands on the queue.

ACTIVATE/DEACTIVATE COMMAND QUEUE

params:
  List of Command Queue Entries (QEs)

Access Command Queue
Invoke Command Queue Maintainer

```
Read Command Queue
Read QEs
Set valid specified QEs to "activated"
  or "deactivated" status
Notify user of any invalid entries
```



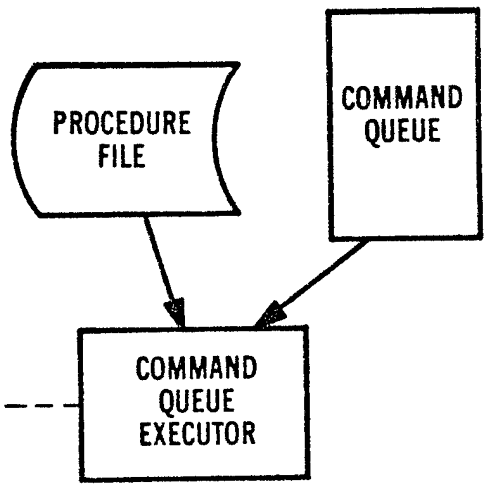
8.4.3.2 Execute Command Queue

The SPF executes each activated command on the queue until all have been executed or a procedure aborts.

EXECUTE COMMAND QUEUE

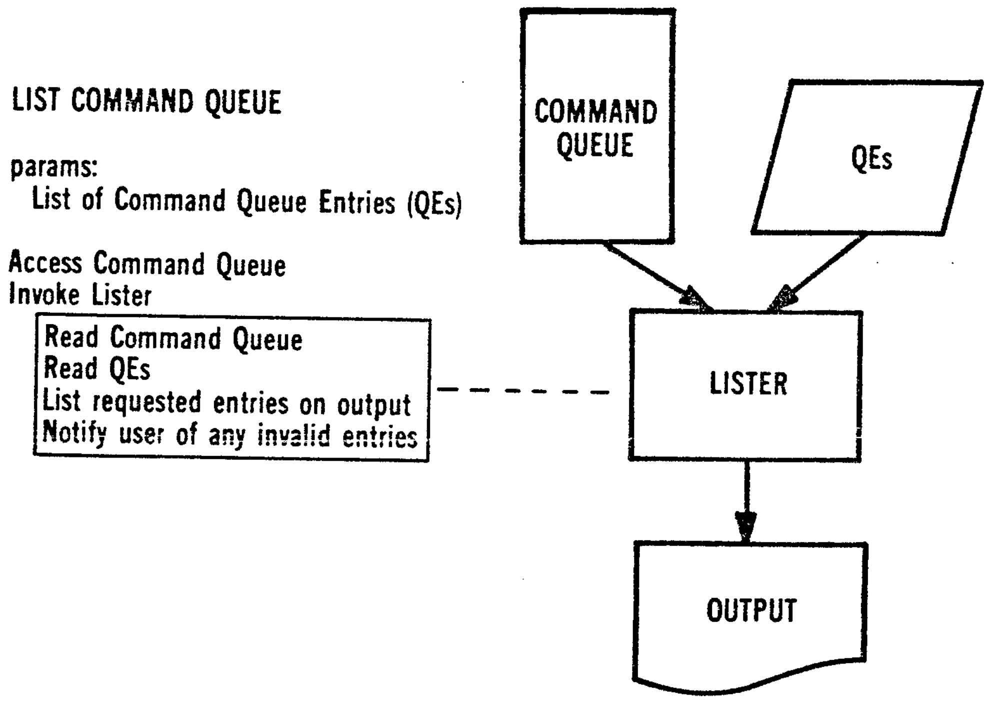
params: (none)

```
Access Command Queue
Invoke Command Queue Executor
    DOWHILE an activated command is on the queue
    • execute next activated command
    ENDDO
```



8.4.3.3 List Command Queue

Each command in the queue is listed with its activation status and the values of all its parameters, whether explicitly specified or supplied by default.



LIST COMMAND QUEUE

params:
List of Command Queue Entries (QEs)

Access Command Queue
Invoke Lister

Read Command Queue
Read QEs
List requested entries on output
Notify user of any invalid entries

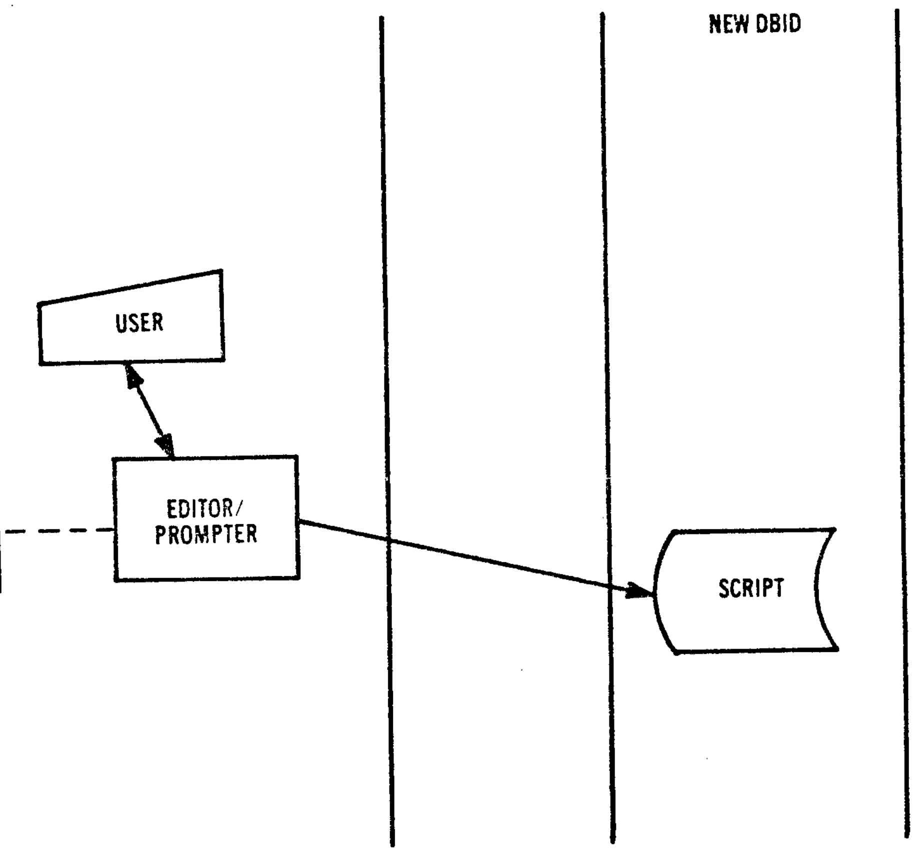
8.4.4 Data Creation Procedures

8.4.4.1 Generate Scripts

The SPF assists the user in preparing scripts for use in other procedures. The scripts are saved on the data base to provide repeatability and accountability.

GENERATE SCRIPTS

```
params:
  Data Base Identifier (DBID)
  Procedure name (PROCNAME)
  Script Name (SCRIPT)
Invoke Editor/Prompter
  DOWHILE no END command received
  • Request input appropriate to PROCNAME
  • Build appropriate script
  ENDDO
Save script
```

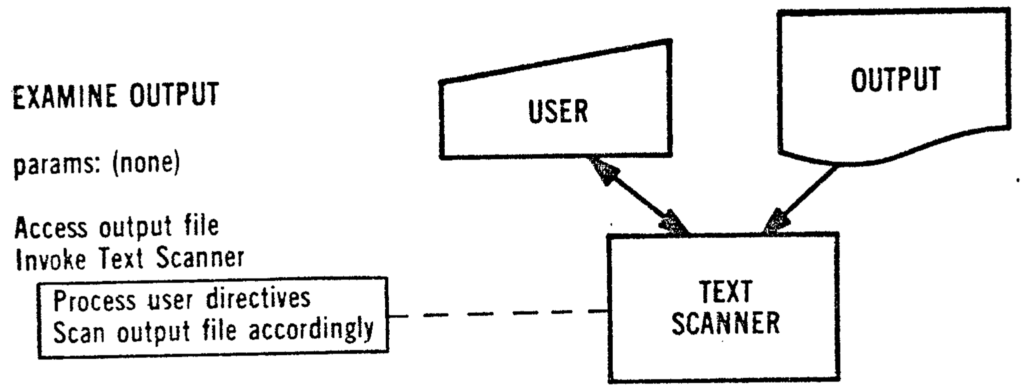


8.4.5 Output Handling Procedures

Output from procedures is collected for later inspection and disposal.

8.4.5.1 Examine Output

This is an interactive procedure to perform operations such as finding all error messages on output without listing the entire output, or determining whether the job output should be printed and/or saved.



```
EXAMINE OUTPUT

params: (none)

Access output file
Invoke Text Scanner
  Process user directives
  Scan output file accordingly
```

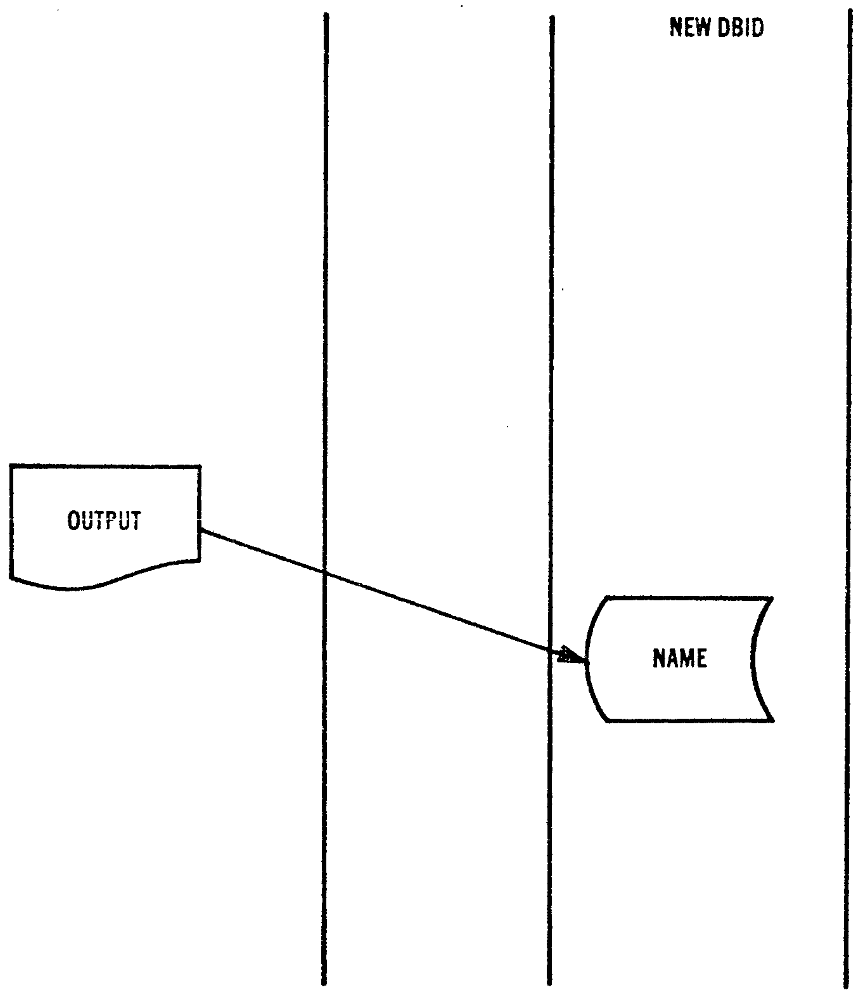
8.4.5.2 Save Output

The user can elect to save an output listing on the data base for purposes such as establishing the success of translations for future audit/report/contract deliverables, or to produce multiple copies of listings.

SAVE OUTPUT

params:
- Name of saved output file (NAME)
- Data Base Identifier (DBID)

Save output file as **NAME** on **DBID**



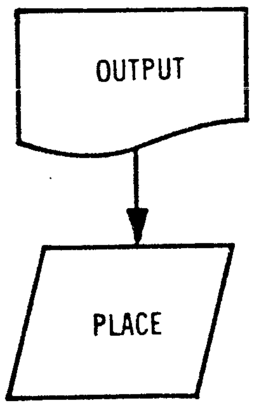
8.4.5.3 Print Output

The user can have the output listing printed on a specified device or deleted.

PRINT OUTPUT

params:
  Identity of printer (PLACE)

Dispose output file to printer



9. VERIFICATION AND VALIDATION PROCEDURE GROUPS

Studies of verification and validation tools are ongoing. Results of these studies are not available at the time of this draft. See Section 7.2, Software Testing Procedures, for related SPF capabilities.

10. QUALITY ASSURANCE PROCEDURE GROUPS

Studies of Quality Assurance tools are ongoing. Results of these studies are not available at the time of this draft. See section 8.2, Software Management Procedures, for related SPF capabilities.

APPENDIX B

SPF PERFORMANCE REQUIREMENTS

The total SPF system includes the host computer and the system software. The system software implements the procedures described in the body of this document. As a system, certain performance characteristics are essential for successful operation; they are: data base capacity, job processing and software tool performance.

Data base capacity includes the projects supported, source and object code, tests, documentation and production information.

Job processing includes the average job load, CPU time and job turnaround time.

Software tool performance includes the time used in processing jobs of varying sizes.

The following data reflect the minimum performance baseline for the SPF.

DATA BASE CAPACITIES

40	PROJECTS
400	ACCOUNTS
1,000	DATA BASES
40,000	MODULES (COMPILABLE MEMBERS)
30,000	INCLUDED MEMBERS
12,000,000	SOURCE LINES
12,000,000	OBJECT WORDS
3,000,000,000	BYTES OF DATA BASE STORAGE

JOB PROCESSING

11,000	JOBS PER MONTH
150	HOURS TOTAL CPU TIME PER MONTH
1	HOUR AVERAGE BATCH JOB TURNAROUND

SOFTWARE TOOL PERFORMANCE

Definition of jobs:

a. The DEVELOP SOFTWARE procedure invokes the CMS-2M Compiler as one of its translators. The procedure edits source code, identifies modified source modules, invokes the translator to produce object code and saves the source and object on the project data base. Timings for typical CMS-2M systems are given below.

(1) Small - 17 CMS-2M source lines

(2) Medium - 468 CMS-2M source lines

(3) Large - 1043 CMS-2M source lines

b. The GENERATE LOAD MODULE/TAPE procedure invokes the SYSGEN 20/14 System Generator to create a load module file or load tape. The job size is determined by the length and the number of elements being loaded, and the resolution of unsatisfied externals. Timings for typical object elements are given below.

(1) SMALL - 5 AYK-14 relocatable object elements with a total length of 30 (hex) words

(2) MEDIUM - 2 AYK-14 relocatable object elements with a total length of 231A (hex) words

(3) LARGE - 11 AYK-14 relocatable object elements with a total length of 2006 (hex) words

c. The EXECUTE TESTS procedure invokes the UYK-20/AYK-14 Computer Emulator to exercise developed object code load modules and save the test results. Timings for typical emulator instruction executions are given below.

(1) SMALL - emulated 500 instructions

(2) MEDIUM - emulated 5000 instructions

(3) LARGE - emulated 50000 instructions

Timings:

CMS-2M Compiler:

JOB	CP TIME	I/O TIME	WALL CLOCK TIME
	sec	sec	min:sec
SMALL	0.161	2.773	00:09
MEDIUM	4.272	26.201	00:51
LARGE	8.818	53.836	01:32

SYSGEN System Generator:

JOB	CP TIME	I/O TIME	WALL CLOCK TIME
	sec	sec	min:sec
SMALL	0.066	6.810	00:13
MEDIUM	0.115	6.851	00:12
LARGE	0.914	13.774	00:22

UYK-20/AYK-14 Emulator:

JOB	CP TIME	I/O TIME	WALL CLOCK TIME
	sec	sec	min:sec
SMALL	0.286	1.687	00:04
MEDIUM	1.796	1.687	00:05
LARGE	26.058	1.686	00:31